VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA STROJNÍ | KATEDRA AUTOMATIZAČNÍ TECHNIKY A ŘÍZENÍ

Řízení systému s využitím signálového procesoru STM32

Control of a System Using Digital Signal Processor STM32

Student: Bc. Jan Šidlo

Vedoucí diplomové práce: Ing. Jolana Škutová, Ph.D.

Ostrava 2020

VŠB - Technická univerzita Ostrava
Fakulta strojní
Katedra automatizační techniky a řízení

Zadání diplomové práce

Student: **Bc. Jan Šidlo**

Studijní program: N2301 Strojní inženýrství

Studijní obor: 3902T004 Automatické řízení a inženýrská informatika

Téma: Řízení systému s využitím signálového procesoru STM32
Control of a System Using Digital Signal Processor STM32

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Seznamte se a popište vývojovou řadu mikrokontrolérů STM32. Zaměřte se na popis vývojové desky STM32-F4-Discovery.
2. Proveďte popis operačních systémů reálného času a analyzujte vývojová prostředí.
3. Popište vybraný laboratorní model pro číslicovou regulaci a zajistěte propojení laboratorního modelu s mikrokontrolérem.
4. Navrhněte a realizujte implementaci číslicového regulátoru pro regulaci laboratorního modelu s využitím mikrokontroléru STM32.
5. Zhodnoťte dosažené výsledky a navrhněte směr dalšího řešení.

Seznam doporučené odborné literatury:

BOBÁL, Vladimír. *Digital self-tuning controllers: algorithms, implementation andapplications.* London: Springer, 2005. ISBN 1-85233-980-2.
FADALI, M. Sami a Antonio VISIOLI. *Digital control engineering: analysis and design.* Second edition. Amsterdam: Academic Press, Elsevier, 2013. ISBN 978-0-12-394391-0.
CHRASCINA, Václav. *Programová podpora periferií mikroprocesorů řady STM32F4.* Ostrava, 2015. Bakalářská práce. Katedra automatizační techniky a řízení, Fakulta strojní, VŠB-Technická univerzita Ostrava. Vedoucí práce Ing. David Fojtík, Ph.D.
VÁŇA, Vladimír. *Mikrokontroléry ATMEL AVR: programování v jazyce C : popis a práce ve vývojovém prostředí CodeVisionAVR C.* Praha: BEN - technická literatura, 2003. ISBN 978-807-3001-025.
WATANABE, Kento. *Introduction to STM32 ARM Microcontroller with STM HAL-Library & SW4STM32* [online]. 1. Kindle, 2017 [cit. 2018-03-02].

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jolana Škutová, Ph.D.**

Datum zadání: 20.12.2019

Datum odevzdání: 18.05.2020

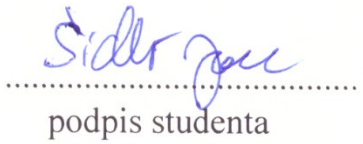
doc. Ing. Renata Wagnerová, Ph.D.
vedoucí katedry

prof. Ing. Ivo Hlavatý, Ph.D.
děkan fakulty

Místopřísežné prohlášení studenta

Prohlašuji, že jsem celou diplomovou práci včetně příloh vypracoval samostatně pod vedením vedoucího diplomové práce a uvedl jsem všechny použité podklady a literaturu.

V Ostravě dne 18. 5. 2020



.....

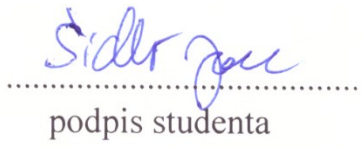
podpis studenta

Prohlášení studenta

Prohlašuji, že:

- jsem si vědom, že na tuto moji závěrečnou diplomovou práci se plně vztahuje zákon č.121/2000 Sb. Zákon o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (dále jen Autorský zákon), zejména §35 (Užití díla v rámci občanských či náboženských obřadů nebo v rámci úředních akcí pořádaných orgány veřejné správy, v rámci školních představení a užití díla školního) a §60 (Školní dílo),
- beru na vědomí, že Vysoká škola báňská –Technická univerzita Ostrava (dále jen „VŠB-TUO“) má právo užít tuto závěrečnou diplomovou práci nekomerčně ke své vnitřní potřebě (§35 odst.3 Autorského zákona),
- bude-li požadováno, jeden výtisk této diplomové práce bude uložen u vedoucího práce,
- s VŠB-TUO, v případě zájmu z její strany, uzavřu licenční smlouvu s oprávněním užít dílo v rozsahu §12 odst.4 Autorského zákona,
- užít toto své dílo, nebo poskytnout licenci k jejímu využití, mohu jen se souhlasem VŠB-TUO, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly VŠB-TUO na vytvoření díla vynaloženy (až do jejich skutečné výše),
- beru na vědomí, že podle zákona č.111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů – že tato diplomová práce bude před obhajobou zveřejněna na pracovišti vedoucího práce, a v elektronické podobě uložena a po obhajobě zveřejněna v Ústřední knihovně VŠB-TUO, a to bez ohledu na výsledek její obhajoby.

V Ostravě dne 18. 5. 2020



podpis studenta

Jméno a příjmení autora práce: Bc. Jan Šidlo

Adresa trvalého pobytu autora práce: Jana Čapka 3098, Frýdek-Místek, 738 01

ANOTACE DIPLOMOVÉ PRÁCE

ŠIDLO, J. *Řízení systému s využitím signálového procesoru STM32*: *diplomová práce*. Ostrava: VŠB – Technická univerzita Ostrava, Fakulta strojní, Katedra automatizační techniky a řízení, 2020, 84 s. Vedoucí práce: Škutová, J.

Diplomová práce se zabývá využitím signálového procesoru z rodiny STM32F7 pro řízením reálného laboratorního modelu. Práce obsahuje přehled řad miktrokontrolérů STM32 s jejich popisem. Detailněji je zaměřena na popis řady STM32F7 a vývojové desky NUCLEO-F767ZI. Je uveden přehled některých dostupných vývojových prostředí. Pro vývoj aplikací je zvoleno prostředí KEIL μVision a kostra programů je zhotovena v aplikaci STM32CubeMX. Důraz je kladen na realizaci a popis základních úloh nezbytných pro řízení reálného laboratorního modelu a zpracování signálu a ke každé je uveden příklad. Zpracované úlohy jsou využití AD převodníku, DA převodníku, ovládání digitální výstupů, čtení stavu digitálních vstupů, práce s čítači, časovači a UART rozhraním. Realizován je program pro číslicový PID regulátor, jehož funkce je ověřena na RC členu a následně je implementován pro řízení otáček laboratorního modelu ventilátoru.

Klíčová slova: KEIL μVision, model ventilátoru, PID regulace, STM32, zpracování signálu

ANNOTATION OF THE MASTER THESIS

ŠIDLO, J. *Control of a System Using Digital Signal Processor STM32: Master theses*. Ostrava: Technical University of Ostrava, Department of Control Systems and Instrumentation, 2020, 84 p. Thesis head: Škutová, J.

Master theses is dealing with control real laboratory model by digital signal processor STM32F7. Theses contains overview of available groups of STM32 microcontrollers with focus to group STM32F7 and development board Nucleo-F767ZI. It is shown example of development environments that are available to use. Chosen is KEIL μVision and program templates are created in STM32CubeMX application. There is created examples of basic applications used at automation control as use of AD and DA controller, control of digital input and outputs, operation with timers, counters and UART. There is created implementation of PID controller and control of laboratory model.

Keywords: KEIL μVision, Fan Model, PID control, STM32, Signal Processing

Obsah

Seznam použitých symbolů

C	kapacita
e	regulační odchylka
$G(s)$	L-přenos (Laplaceův přenos)
G_R	přenos regulátoru
G_S	přenos regulované soustavy
k	relativní diskrétní čas
k_p	zesílení regulátoru
R	odpor
s	komplexní proměnná, nezávisle proměnná u obrazu v L-transformaci
t	(spojitý) čas
t_R	doba regulace
T	vzorkovací perioda
T_d	dopravní zpoždění u spojitých systémů
T_D	derivační časová konstanta
T_I	integrační časová konstanta
T_1	setrvačná časová konstanta
T_w	(požadovaná) časová konstanta uzavřeného regulačního obvodu, ladicí parametr
u	akční veličina, řízení
U_i	elektrické napětí
v	poruchová veličina
w	žádaná veličina
y	regulovaná veličina, výstupní veličina
τ	časová konstanta

Seznam použitých zkratek

API	rozhraní pro programování aplikací (Application Programing Interface)
BLE	Bluetooth s nízkými nároky na spotřebu (Bluetooth Low Energy)
CAN	sběrnice, využívaná nejčastěji v automobilech (Controller Area Network)
CMSIS	sada knihoven usnadňující programování mikroprocesorů (Cortex Microcontroller Software Interface Standard)
CRC	kontrolní součet (Cyclic Redundancy Check)
CoreMark	test pro měření výkonosti
DSP	digitální signálový procesor (Digital Signal Processor)
ETH	standardizovaná technologie pro realizaci počítačových sítí (Ethernet)
FMC	rozhraní pro komunikaci s externí pamětí (Flexible Memory Controller)
FPU	matematický koprocesor (Floating-Point Unit)
GPIO	univerzální vstup/výstup (General Purpose Input/Output)
HAL	knihovna pro usnadnění práce s periferiema (Hardware Abstraction Layer)
HSI	vysokorychlostní oscilátor (High-Speed Internal)
I^2C	sériové komunikační rozhraní (Inter-Integrated Circuit)
IEEE	standardizační agentura (Institute of Electrical and Electronics Engineers)
IWDG	ochrana proti zacyklení (Independent Watchdog)
LED	svítivá dioda (Light-Emitting Diode)
LPTIM	časovač s nízkým příkonem (Low-Power Timer unit)
MCU	mikroprocesor (MicroController Unit)
NVIC	metoda prioritizace přerušení (Nested Vector Interrupt Control)
NVM	dhuh paměti (Non-Volatile Memory)
PLL	fázový závěs (Phase Lock Loop)
RCC	periférie mikroprocesoru (Reset and Clock Control)
RF	vysokofrekvenční (Radio Frequency)
RISC	architektura mikroprocesorů (Reduced Instruction Set)
RTC	jednotka reálného času (Real Time Clock)
SAI	sériový interface pro zpracování zvuku (Serial Audio Interface)
SDIO	rozhraní pro připojení paměťové karty (Secure Digital Input Output)

UART	rozhraní pro sériovou komunikaci (Universal Asynchronous Receiver/Transmitter)
ULPMark-CP	test pro měření výkonosti
V/V	Vstupně-Výstupní
WWDG	ochrana proti zacyklení (Window WatchDoG)

Úvod

Řízení a automatizace procesů je neoddělitelně spojené s PLC automaty, které mohou být pro realizaci nevhodné z důvodu rozměrů, finanční náročnosti či programového vybavení. Z tohoto důvodu je vhodné se uchýlit k realizaci řízení procesu pomocí výkonných mikrokontrolérů. Ty v dnešní době mají v jednom pouzdře integrovanou řadu komponent, které je třeba k této aplikaci využít.

Vývoj mikroelektrotechniky a miniaturizace nabízí mikrokontroléry minimálních rozměrů, které se tím stávají pro použití v praxi mnohem zajímavější. Další výhodou je, že se už nejedná pouze o zařízení obsahující pouze digitální vstupy a výstupy, avšak obsahují hodně různých rozhraní, periferie a pomocnými procesory. Mezi rozhraními můžeme najit od analogových vstupů a výstupů, CAN, I2C, SPIO, až po Bleuetooth, Ethernet či WiFi. Pokud se podíváme na periferie, budou to třeba časovače, čítače a DMA. Obsahuje také specializovaný matematický koprocesor, určený pro výpočet operací s pohyblivou desetinnou čárkou a digitální signálový procesor pro větší rychlost zpracování signálu.

Mikrokontroléry společnosti STMicroelectronics jsou v současnosti jedny z nejrozšířenějších a nejpoužívanějších. Vyrábějí se v několika úrovních vybavenosti a tím je možné zvolit, nejvhodnější model pro konkrétní reálné nasazení.

V dnešní době je častým požadavkem maximalizovat možnosti navrženého řešení technických prostředků a současně minimalizovat náklady na jejich pořizovací cenu, a dále také je neméně důležitým požadavkem, aby výsledky regulace byly dosaženy s maximální možnou kvalitou.

Výhodou použití mikrokontrolérů je mimo jiné jejich implementace v blízké vzdálenosti senzorům a při získání signálů z těchto technických prostředků je zaručeno menší rušení signálů než u přenosu analogového signálu ke vzdálenému počítači pro další zpracování..

1 Vývojové řady mikrokontroléru STM32

STM32 od společnosti STMicroelectronics je vedoucím zástupcem Arm Cortex-M mikrokontrolérů. Jedná se 32bitové RISC procesory postavené na konceptu společnosti Arm holding. Rodina STM32 mikrokontrolérů je rozdělena na 12 podskupin, každá s vlastními parametry. [STMicroelectronics, 2018c]

1.1 Rozdělení vývojové řady STM32 podle požadovaných vlastností

- Vysoký výkon - vysoký stupeň integrace a obsáhlá základna periferií
    - **STM32H7**: nejvýkonnější řada STM32 mikroprocesorů s pokročilými funkcemi včetně DSP a FPU instrukcí postavená na architektuře Cortex-M7 s 1 až 2 MB paměti flash (2020 CoreMark),
    - **STM32F7**: řada velice výkonných mikroprocesorů s pokročilými funkcemi včetně DSP a FPU instrukcí postavená na architektuře Cortex-M7 s 256 kB až 2 MB paměti flash (1082 CoreMark),
    - **STM32F4**: řada výkonných mikroprocesorů s pokročilými funkcemi včetně DSP a FPU instrukcí postavená na architektuře Cortex-M4 s 64 kB až 2 MB paměti flash (608 CoreMark),
    - **STM32F2**: střední řada mikroprocesorů s vynikajícím poměrem cena/výkon, postavená na architektuře Cortex-M3 s 64 kB až 2 MB paměti flash (398 CoreMark)
- Nejžádanější – škálovatelný balík MCU pro velké spektrum aplikací
    - **STM32F3**: vylepšená řad F1 s různou úrovní pokročilých analogových periferií, postavena na architektuře Cortex-M4 s 16 to 512 kB paměti flash (245 CoreMark),
    - **STM32F1**: základní řad postavena na architektuře Cortex-M3 s 16 kB až 1 MB flash paměti (108 Cormark),
    - **STM32F0**: nejjednodušší řada obsahující 8-/16-bitové mikroprocesory postavené na architektuře Cortex-M0 s 16 až 256 kB paměti flash (105 CoreMark).

- Ultra nízká spotřeba – aplikace s nízkými příkonovými nároky
    - **STM32L4+**: mikroprocesory s ultra nízkou spotřebou s vyšším výkonem, postavené na architektuře Cortex-M4 s 1 až 2 MB paměti flash (233 ULPMark-CP / 55 ULPMark-PP / 410 CoreMark),
    - **STM32L4**: nejlepší ve třídě ultra nízké spotřeby s vyšším výkonem, postavené na Cortex-M4 s 128 kB až 1 MB paměti flash (347 ULPMark-CP / 121 ULPMark-PP / 273 CoreMark),
    - **STM32L1**: správná volba pro 32-bit aplikace, postavena na architektuře Cortex-M3 s 32 až 512 kB paměti flash (81 ULPMark-CP / 93 CoreMark),
    - **STM32L0**: správná volba pro 8-/16-bitové aplikace, postaveno na architektuře Cortex-M0+ s 8 až 192 kB paměti flash (244 ULPMark-CP / 95-ULPMark-PP / 75 CoreMark).
- Bezdrátové – multiprotokolové vysílače s přijímači (transceivery) v pásmu 2,4 GHz a ultra nízkou spotřebou
    - **STM32WB**: dvojjádrová (Cortex-M4/M0+) architektura (216 CoreMark) podporující bezdrátovou technologii ve standardu 5.0 a standard pro přenos dat (IEEE 802.15.4). Vysokofrekvenční část s výstupním výkon 6 dBm a citlivost přijímače
      -96 dBm / -100 dBm (BLE / IEEE 802.15.4). Pamět flash od 256 kB do 1 MB.

1.2 Podskupina STM32F7

Jedná se o řadu vysoce výkonných mikrokontrolérů s DSP a FPU instrukcemi, která je založena na ARM Cortex-M7 architektuře s ST NVM technologii a Chrome ART akcelerátorem. Díky kombinaci těchto technologii dosahuje nejvyšší výsledky v testech výkonu, a to až 462DMIPS/1082 CoreMark s flash pamětí na frekvenci až 216 MHz. [STMicroelectronics, 2018d]

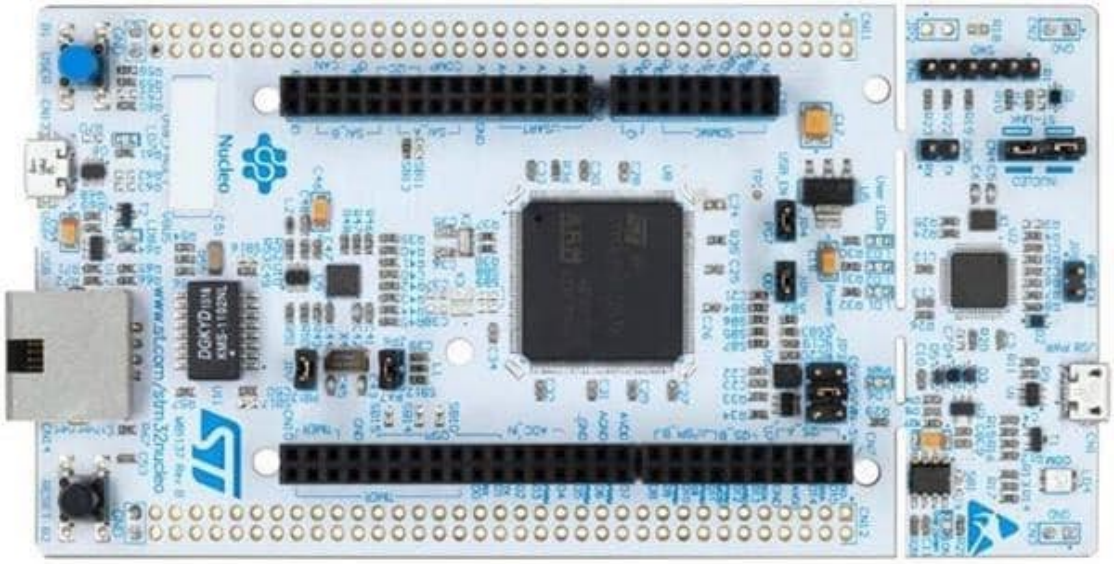
Vlastnosti řady STM32F7:

- AXI a vícenásobná AHB matice sběrnic pro propojení jádra, periférii a paměti,
- až 16 kB + 16 kB I-cache a D-cache,
- až 2 MB vestavěné flash paměti,
- dva obecné DMA řadiče a jeden DMA řadič vyhrazený pro Ethernet rozhraní, vysokorychlostní USB-OTG a Chrom-ART akceleraci grafických operací,

- rychlost periferií nezávislou na rychlosti jádra,
- více periferií jako např. dvě SAI (serial audio interface) s podporou SPDIF výstupu, tři I^2S half-duplex interface s SPDIF vstupem, dvě USB-OTG rozhraní s vyhrazeným napájecím zdrojem a dvojité a čtyřnásobné SPI rozhraní pro připojení flash paměti,
- velká SRAM - až 512 kB paměti pro data.

2 Vývojová deska Nucleo-STM32F767ZI

Pro seznámení se s určitým typem mikrokontroléru a usnadnění vývoje je lepší nepracovat hned ze začátku s prototypem zařízení, ale je lepší využít vývojovou desku od výrobce mikrokontroléru. Vývojová deska umožnuje snadný vývoj aplikace v testovacím prostředí a následný bezproblémový přechod na produkční zařízení. [STMicroelectronics, 2018f]



Obr. 1 - Vývojový deska Nucleo-STM32F767ZI [STMicroelectronics, 2019].

Vlastnosti vývojové desky Nucleo-STM32F767ZI

Neocenitelnou podporu pro vývoj jakékoliv aplikace poskytne vývojová deska. Tento konkrétní model je osazený jedním z nejvýkonnějších mikrokontrolérů společnosti STMicroelectronics.

Vývojová deska má tyto vlastnosti:

- možnosti připojení rozšíření Arduino Uno rev. 3 a STMicroelectronics Morpho a Zio s plným přístupem ke všem STM32 I/O,
- integrované ST-LINK/V2-1 rozhraní s SWD konektorem, pro ladění a programování mikrokontroléru,
- možnost napájení desky z USB VBUS nebo z externího zdroje (3,3 V, 5 V, 7-12 V),
- 3 uživatelsky ovladatelné LED diody,
- dvě tlačítka – uživatelské a RESET,
- USB-OTG interface,
- Ethernet rozhraní.

2.1 Mikrokontrolér STM32F767ZIT6

Jedná se o zástupce 32bitové Arm Cortex-M7 rodiny, rozšířený o FPU, MPU a adaptivní real-time akcelerátor (Art Accelerator) s těmito vlastnostmi [STMicroelectronics, 2019b]:

- mikroprocesor STM32F767ZIT6 v pouzdře LQFP144,
- maximální frekvence CPU 216 MHz,
- 2 MB flash,
- 512 kB statická RAM paměti,
- 114 univerzálních vstupně/výstupních pin s možností externího přerušení,
- tři 24 kanálové 12-bitové AD převodníky,
- dva DA převodníky s rozlišením 12 bitů,
- časovače:
    - 10 obecných,
    - 2 rozšířené,
    - 2 základní,
    - s nízkým příkonem,
- rozhraní:
    - 8x USART/UART,
    - 4x I^2C,
    - 6x SPI,
    - 2x SAI,
    - SDMMC,
    - kamera,
    - LCD-TFT displej,
    - 4x SPDIF-RX,
    - USB 2.0 OTG HS/FS,
- 2 hlídací časovače,
- 3 sběrnice CAN 2.0 B v aktivní variantě,
- generátor náhodných čísel.

Tyto vlastnosti umožnují mikrokontroléru velice široké uplatnění a využití pro různé aplikace.

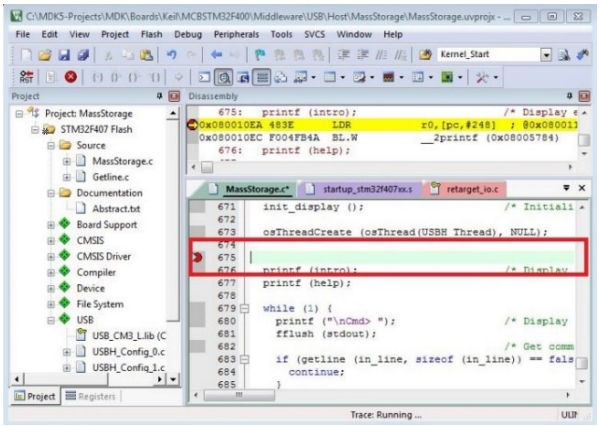
3 Vývojová prostředí pro STM32

Jedná se o aplikace, která programátorovi usnadňuje práci. Většinou neobsahuje pouze editor zdrojového kódu, ale integruje i další funkcionality, jako jsou třeba volání kompilátoru kódu se zobrazením výstupu a znázorněním případné chyby, možnost ladění aplikace, případně vizualizaci obsahu proměnných programu běžícího v mikrokontroléru.

Detailněji se podíváme na vývojová prostředí KEIL µVision, Atollic TrueSTUDIO, STM32Cube IDE a System Workbench for STM32.

3.1 KEIL µVision

Vývojové prostředí µVision IDE v sobě integruje správu celého projektu, editor zdrojového kódu a debugger. Je to jednoduché prostředí, které urychlí vývoj. µVision Debugger poskytuje jednotné prostředí, ve kterém je možné vyvíjenou aplikaci otestovat, odladit případné chyby a optimalizovat. Lze využít zarážky v kódu, nahlížení na hodnoty proměnných, řízené spuštění kódu a plný náhled na stav periférií MCU.



Obr. 2 - Vývojové prostředí KEIL µVision.

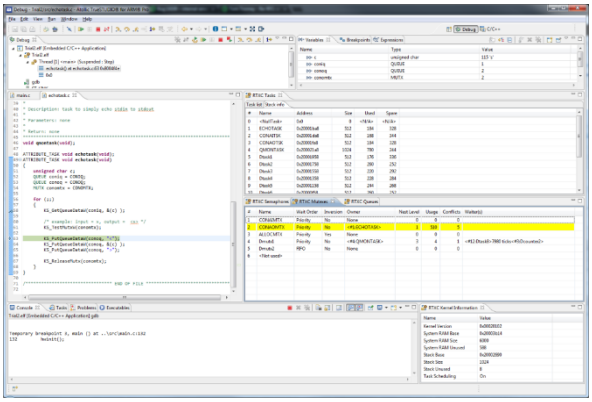
Toto prostředí v licenci MDK-Lite, která je volně dostupná, má následující omezení:

- Programy, které vygenerují více než 32 kB kódu a dat, nebudou zkompilovány ani slinkovány.
- Ladící modul podporuje pouze programy menší než 32kB.
- Kompilátor a linker negeneruje nezávislý kód a data. Přepínače apcs /ropi /rwpi /pic/ pid kompilátoru a linkeru jsou vypnuté.

- Kompilátor vytváří objekty v Symbolic Output Format, které nemohou být slinkovány linkery třetích stran. Plně licencovaný kompilátor generuje ELF/DWARF soubory, které jsou plně podporovány linkery třetích stran.

3.2 Atollic TrueSTUDIO

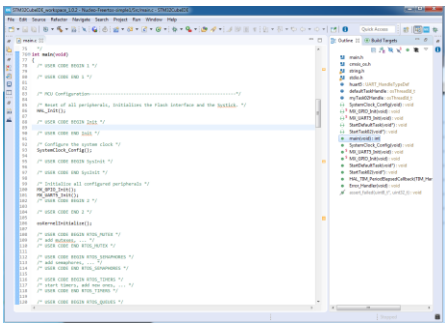
Jedná se o vývojové prostředí vyvíjené společností Atollic, kterou koupila společnost STMicroelectronics. Je dostupný pro komerční i nekomerční užití. Obsahuje vysoce optimalizující C/C++ překladač a sestavovací program. Součástí je i projekt management, pro přehledné zobrazení všech připojených souborů a knihoven. Vývojové prostředí umožnuje pracovat se softwarem pro správu verzí SVN, pro kontrolu změn kódu. Prostředí dokáže zobrazit využití RAM a flash paměti na základě poslední kompilace – celkovou velikost, využitou část a volnou část. Podporuje sondy pro odhalování chyb jako ST-link, SEGGER nebo P&E micro. V prostředí je možno provést analýzu pádů programu – jak, proč a kde k nim došlo.



Obr. 3 - Vývojové prostředí Atollic TrueSTUDIO.

3.3 STM32Cube IDE

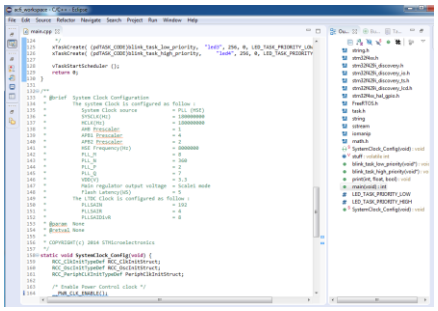
Jedná se o vývojové prostředí vyvíjené přímo společností STMicroelectronics. Je postavené na volně dostupném prostředí Eclipse a vyvíjené v jazyce Java. Obsahuje editor kódu a správce projektu. Je možné z něj přímo vyvolat kompilování kódu, sestavení programu a nahrát program do mikrokontroléru. Prostředí má v sobě integrovaný nástroj STM32CubeMX, který umožnuje sestavit kostru programu a přednastavit periferii, které se rozhodneme využívat. Bohužel nemá integrované žádné rozšiřující knihovny, avšak je k získání bezplatně.



Obr. 4 - Vývojové prostředí STM32Cube IDE.

3.4 System Workbench for STM32

Prostředí Systém Workbench for STM32 je vyvíjeno společností AC6, kterou koupila společnost STMicroelectronics. Základem tohoto prostředí je volně dostupné vývojové prostředí Eclipse. Napsané je v jazyce Java. Obsahuje editor kódu a správce projektu. Je možné z něj přímo vyvolat kompilování kódu, sestavení programu a nahrát program do mikrokontroléru. Stejně jako STM32Cube IDE umožnuje zkompilovat kód, sestavit program a odeslat do procesoru. Výsledný program zle odladit. Oproti STM32Cube IDE neobsahuje integrovaný nástroj STM32CubeMX. Jinak se jeví naprosto totožně.



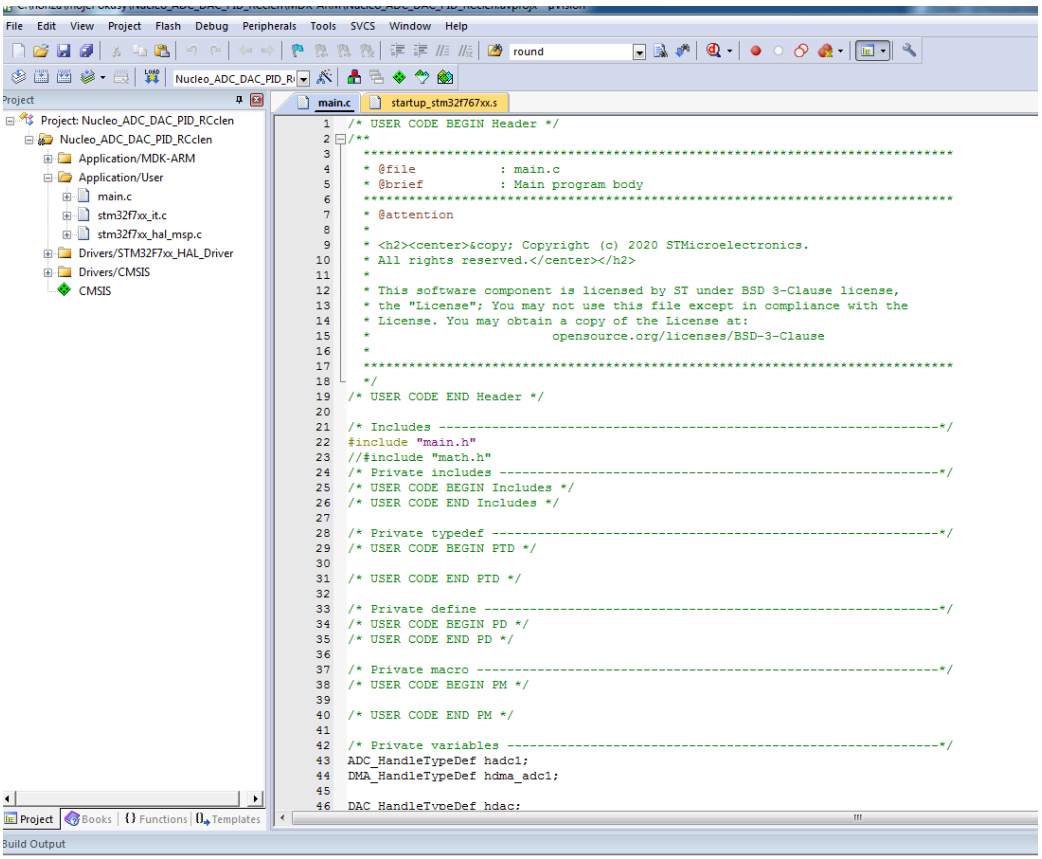
Obr. 5 - Prostředí System Workbench for STM32.

3.5 Volba prostředí

Z plejády možných prostředí, které jsou na trhu k dispozici, jsem si zvolil produkt společnosti Keil pro jeho přehlednost prostředí i vybavenost. Vlastní prostředí má přehledný modul pro ladění s možností napojení mikrokontroléru pomocí ST-LINK. Výhodou je, že lze použít kostru programu vygenerovanou utilitou z dílny STMicroelectronics s názvem STM32CubeMX, která umožňuje pohodlné připravení programu a nastavení jednotlivých periferií.

3.6 Orientace v prostředí KEIL µVision IDE

Následující popis orientace v prostředí µVision bude popisován pro verzi 5.29. Na Obr. 6 je zobrazeno prostředí KEIL µVision IDE. Je vertikálně rozděleno do dvou sekcí. Levá část je orientace v adresářové struktuře projektu a pravá je obsah souboru vybraného k editaci.

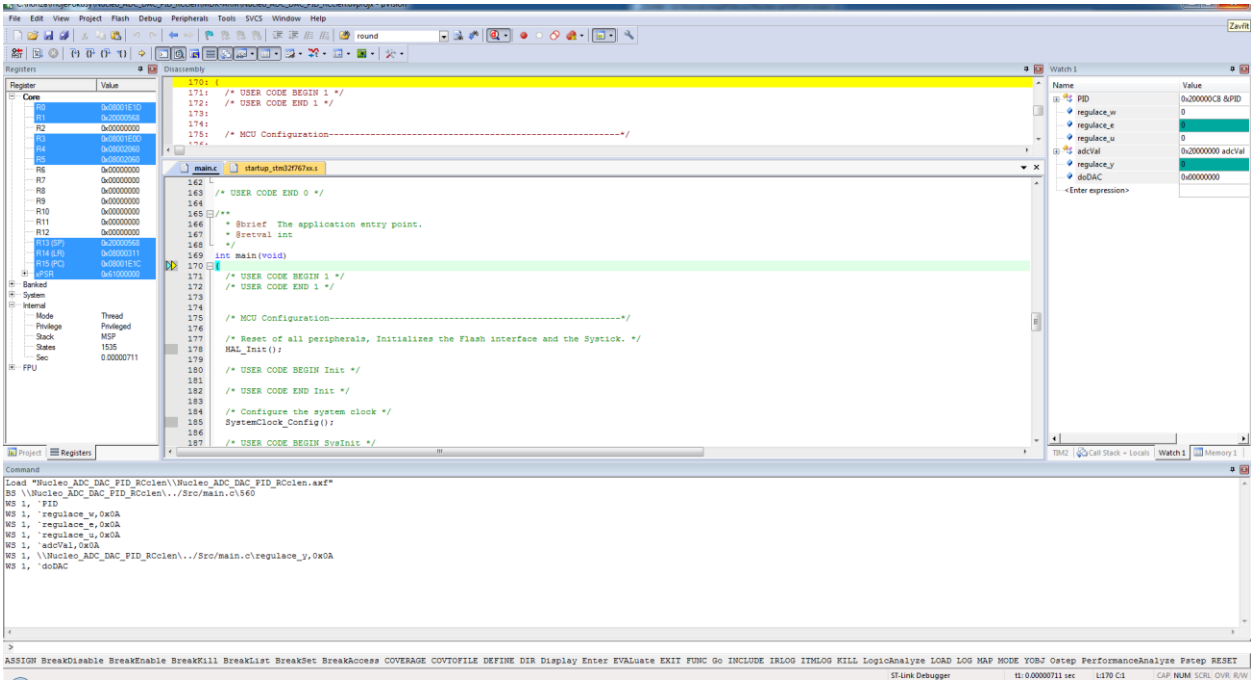


Obr. 6 - Pracovní prostředí KEIL µVision 5.

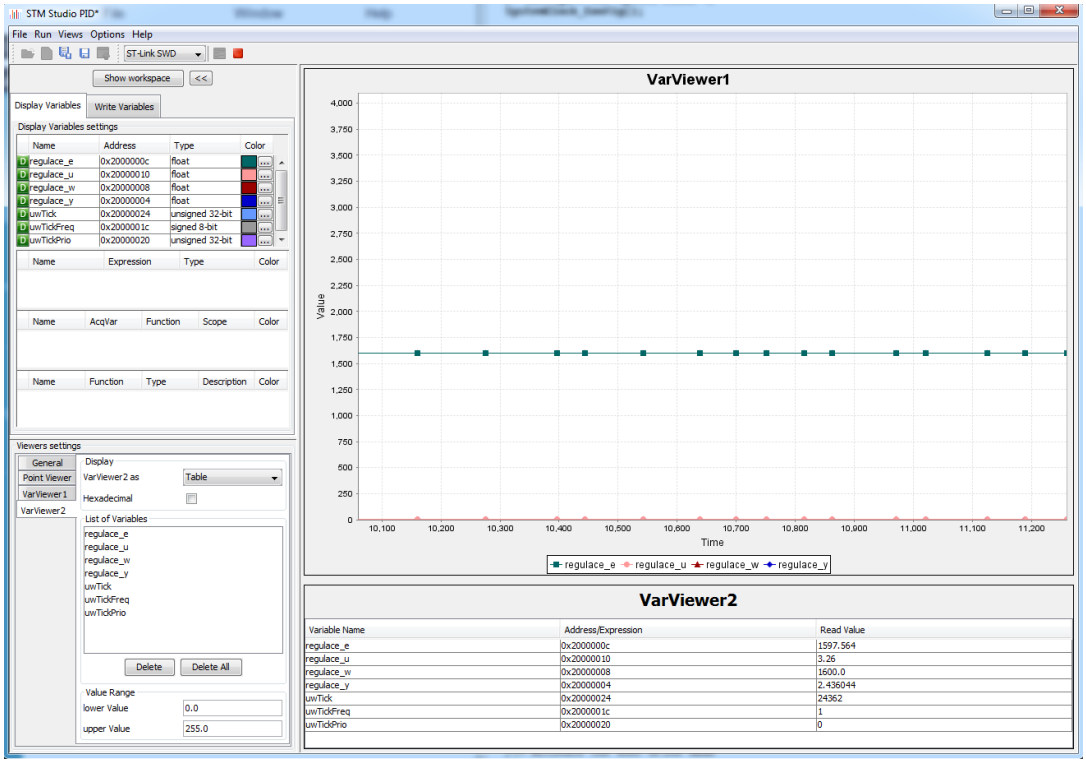
Překlad a sestavení programu do binární podoby provedeme pomocí tlačítka nebo z menu Project->Build Target. Průběh akce je možné sledovat ve spodní části okna v sekci „Build Output“. Zde jsou zobrazeny i případné chyby, které je třeba opravit a celou akci zopakovat.

Nahrát program do mikroprocesoru lze pomocí tlačítka nebo z menu Flash->Download. Po úspěšném nahrání programu do mikroprocesoru je potřeba jej restartovat, ať se začne vykonávat nový program.

Pro sledování chování a ladění programu je potřeba se přepnout do sekce ladění programu pomocí tlačítka nebo z menu Debug->Start/Stop Debug Session. Proces ladění programu je zřejmý z Obr. 7.



Obr. 7 - Okno ladění programu prostředí KEIL µVision 5.



Obr. 8 - Okno programu STM Studio.

V módu ladění programu je možné do okna nahlížení do obsahu proměnných „Watch“ vložit globální proměnné, které je potřebné monitorovat při následném krokování nebo běhu programu sledovat jejich změny.

Ladění programu v prostředí KEIL µVision 5 ve spojení s deskou Nucleo nedovoluje vykreslování průběhu proměnných, které se často hodí a má lepší vypovídající hodnotu a umožní lepší představu o průběhu a vývoji sledovaných proměnných. Z toho důvodu je vhodné použít program od STMicroeletronics s názvem STM Studio. Ten je volně dostupný na stránkách společnosti.

V programu STM Studio je možné vidět proměnné v třech zobrazeních – graf, sloupcový graf a tabulka. Tyto zobrazení lze použít i současně – viz Obr. 8.

4 Operační systémy reálného času

Jedná se o více úlohový (multitasking) operační systém zajišťující prostředí pro provoz aplikací pracujících v reálném čase, které zpracovávají data nebo řídí technologický proces. Operační systémy reálného času (Real-Time Operating System, RTOS) nabízí oproti běžným OS záruku, že na aplikaci běžící v rámci více úlohového zpracování dojde řada ve zpracovávání přesně v požadovaném čase. Úlohy v těchto systémech mají vymezený určitý časový úsek, během kterého mohou být vykonávány - deadline. Ani RTOS nedokážou zajistit načasování úloh na 100 %, ale snaží se tomuto cíli maximálně přiblížit. Podle míry striktnosti se dělí na dva typy Hard RTOS a Soft RTOS. [Srovnal, 2003]

4.1 Typy RTOS

RTOS se dají rozdělit do dvou skupin Hard RTOS a Soft RTOS. Rozdělení do těchto skupin je dáno striktností k dodržení času, kdy dojde k předání kontroly dalšímu procesu.

Hard RTOS

Tento typ RTOS dodržuje časování úloh striktně a čas předání kontroly dalšímu procesu je absolutní. RTOS užívají k rozdělení zdrojů běžícím procesům (zejména procesorového času) speciální plánovače úloh (process scheduler). U RTOS se očekávají tyto vlastnosti:

- preemptivní plánovač,
- velký počet nastavitelných priorit vláken,
- přesné hodiny reálného času.

Vlastnosti plánovačů procesů:

- minimální latence při reakci na událost a při přepínání vláken,
- minimalizace času, kdy jsou zakázané přerušení,
- preemptivní plánování založené na prioritách.

Nejčastěji používané algoritmy plánovačů:

- Round-robin – každému procesu je přiřazen stejný čas, během kterého může být vykonáván. Po uplynutí tohoto času se přejde na další proces.
- Preemptivní plánování – každý proces dostane při svém založení prioritu. Tato vlastnost určuje, s jakou vahou bude vykonáván.
- Kombinace round-robin a preemptivního plánování.

Soft RTOS

Typ Soft RTOS není tak striktní v dodržování času předání kontroly dalšímu procesu a dovoluje drobné odchylky.

4.2 Vybrané implementace RTOS

Prostředí systémů reálného času pro souběžný běh více úloh spravedlivě a rovnoměrně rozloží čas procesoru mezi ně. Na trhu jich je spousta, a proto budou zmíněny jen některé.

4.2.1 Operační systém Keil CMSIS-RTOS RTX

Jedná se o produkt vyvíjený společností Keil. Je k dispozici bezplatně včetně zdrojových kódů. Obsahuje flexibilní plánovače procesů (roudn-robin, pre-emptive a collaborative). Umožnuje provádět vysokorychlostní operace v reálném čase s nízkým zpožděním přerušení, má možnost mít neomezený počet procesů s 254 prioritami a může provozovat neomezený počet mailboxů, semaforů, mutexů a časovačů. Dále podporuje multithreading a thread-safe operace, je v něm zabudovaná podpora pro ladění a podpora rozhraní CMSIS-RTOS v2.

4.2.2 Operační systém FreeRTOS

Implementace tohoto typu software je jedna z nejrozšířenějších. Podporuje MCU od více než 17 výrobců. Operační systém je vyvíjen pod MIT licencí, což zajištuje jeho bezplatnou dostupnost i pro komerční aplikace. Je velice jednoduchý, škálovatelný a jednoduchý k používání. Jeho kód je rozdělen do 3 souborů. Výhodou uživatele jsou šablony příkladů pro každou podporovanou platformu. V tomto RTOS je integrováno rozhraní CMSIS-RTOS v2.

4.2.3 ChibiOS/RT

Vysoce výkonným open-source operačním systémem reálného času je ChibiOS/RT, který byl vyvinut s cílem, ať vynikne svým výkonem a velikostí kódu. Je plně statický, má malé nároky na paměť a silnou podporu pro odlaďování chyb. Tento typ OS je plně kompatibilní s HAL OSAL a MISRA 2012, včetně podpory rozhraní CMSIS-RTOS v2.

4.3 Výběr implementace RTOS

Z uvedeného přehledu operačních systémů reálného času měl protekční postavení FreeRTOS, který je integrovaný do programu STM32CubeMX, zmíněného v kapitole 3.3. V tomto nástroji je možné provést změnu jeho nastavení a přípravu na integraci s hardwarovými částmi mikrokontroléru STM32F767.

5 Realizace testovacích rozhraní STM32

Pro implementaci řízení procesu v mikrokontroléru je zapotřebí zvládnout několik typů úloh:

- AD převodník – mikrokontrolér je digitální zařízení a spousta senzorů v běžném světě je analogových a tak je nutné pro další zpracování převést analogový signál na digitální.
- DA převodník – obdobně jako u AD převodníku je zapotřebí mít inverzní nástroj, který zajistí možnost generování spojitých signálů.
- Sériová linka – pro komunikaci s mikrokontrolérem je vhodné mít k dispozici nástroj, který to umožňuje. Nejjednodušší pro implementaci je obousměrná sériová komunikace, pomocí které lze zadávat povely pro mikrokontrolér a taktéž posílat uživateli diagnostické informace či data z probíhajícího dědí v mikroprocesoru.
- Časovač – vhodný nástroj nejen pro měření časových úseku, ale i počítání pulzu určitý čas. Pomocí časovače je možno ovládat i AD převodník a tím jej spouštět v pevně daný čas, tak ať je možné převádět analogovou hodnotu s pevnou periodou snímání.

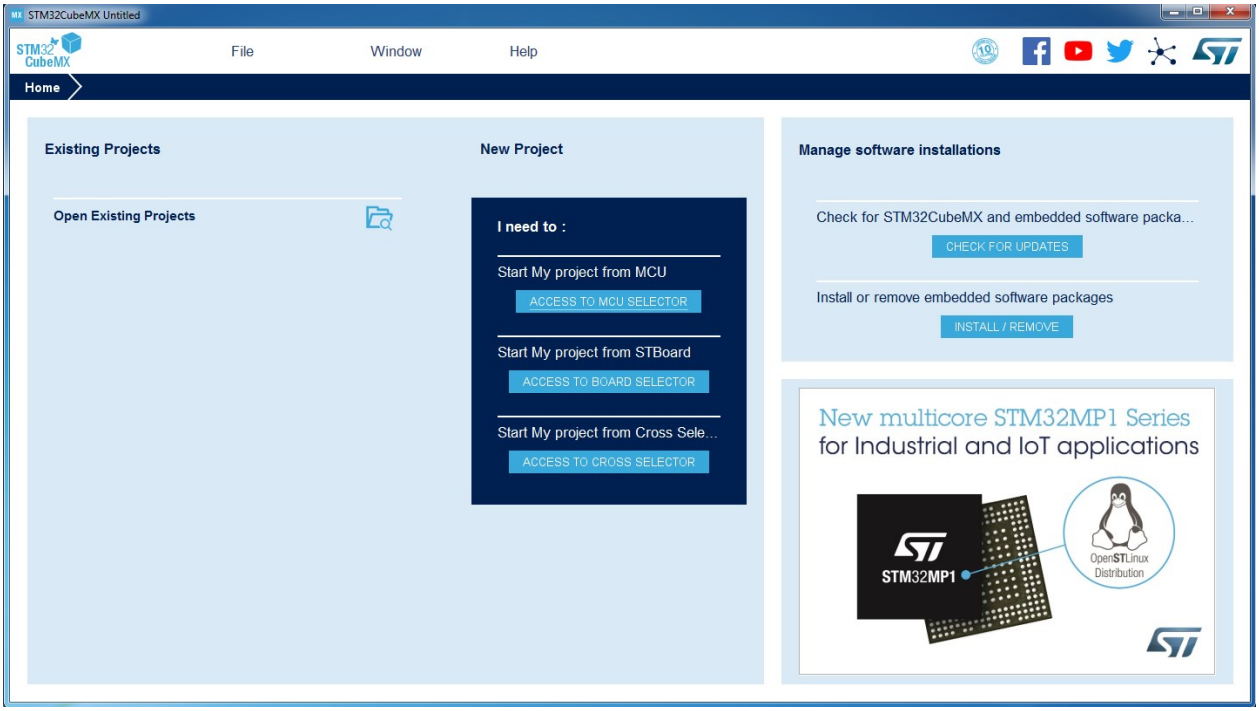
5.1 Konstrukce kostry programu pro STM32

K přípravě programu, nastavení potřebných periferií a připojení potřebných knihoven je nápomocna utilita s portfolia STMicroelectronics s názvem STM32CubeMX, která je volně dostupná na stránkách společnosti. V době psaní práce je k dispozici verze 5.5.0 programu STM32CubeMX.

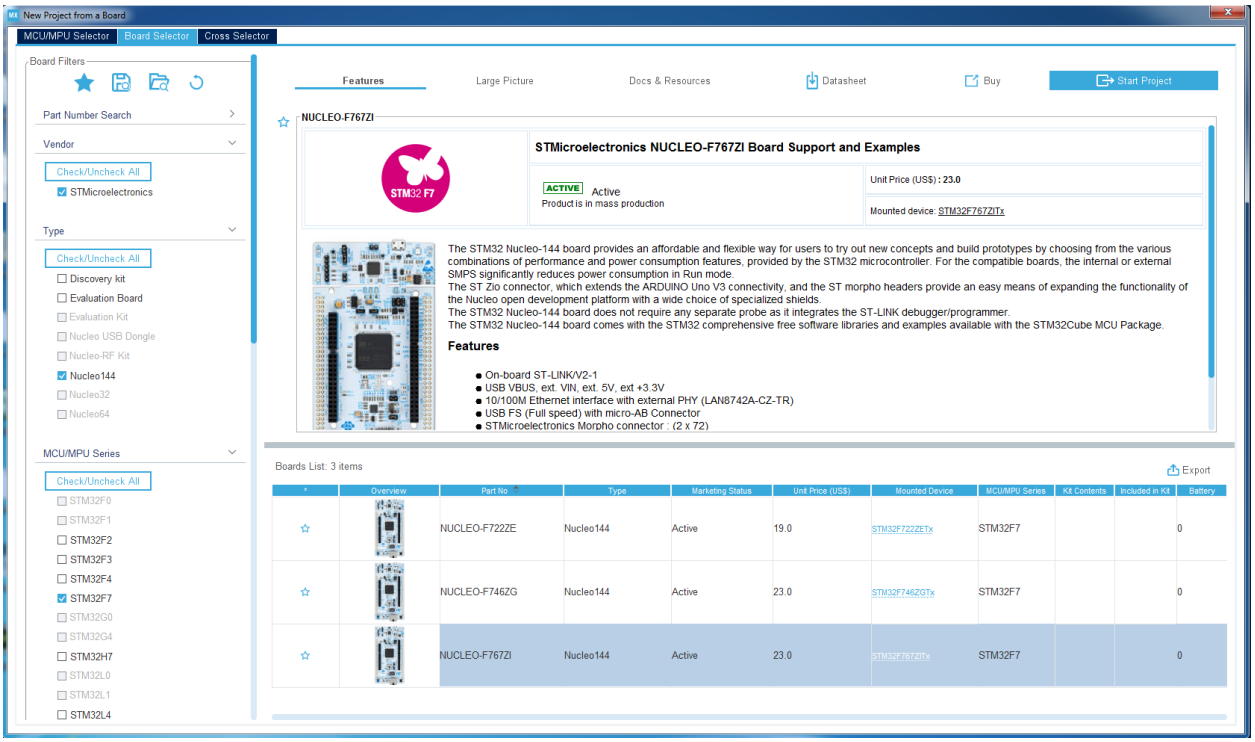
V STM32CubeMX je možné začít projekt výběrem na základě použitého procesoru, vývojové desky nebo na základě požadovaných parametrů – viz Obr. 9.

V následujících příkladech bude použit výběr na základě vývojové desky („Board Selector“). Použitá vývojová deska je ST Nucleo-F767ZI, „Vendor“ (výrobce): STMicroeletronics. Type (typ): Nucleo144, MCU/MPU Series (řada mikrokontroléru): STM32F7 – Obr. 10.

Výběr lze potvrdit tlačítkem „Start Project“ a následně na otázku „Initialize all peripherals with their default Mode?“ (provést inicializaci všech periferií v jejich obvyklém modu) se zvolí „No“. Tímto je zaručeno, že uživatel si volí je ty funkcionality desky, které aktuálně bude používat.



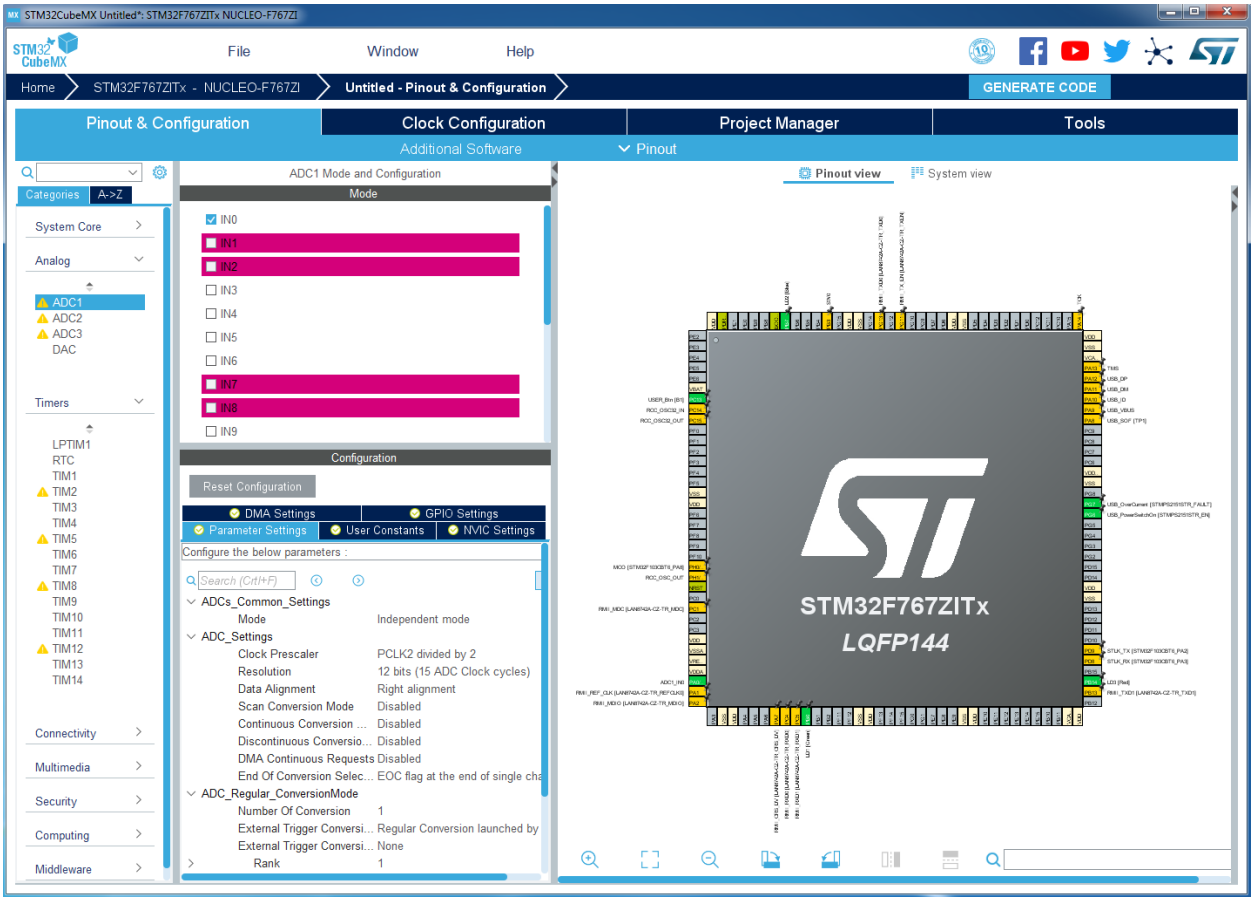
Obr. 9 - Úvodní obrazovka programu STM32CubeMX.



Obr. 10 - Výběr vývojové desky v prostředí STM32CubeMX.

Orientace v programu je poměrně intuitivní. V sekci „Pinout & Configuration“ (rozvržení nožiček a konfigurace) je 8 sekcí, do kterých jsou rozděleny periferie vývojové desky:

- System Core – systémová nastavení procesoru
  - CORTEX_M7 – nastavení flash interface, ART accelerator, Instruction Prefetch, CPU ICache/DCache, jednotky Cortex Memory Protectio,
  - DMA – nastavení přímého zápisu do paměti z periferií,
  - GPIO – nastavení jednotlivých nožek mikrokontroléru, pojmenování, nastavení vlastností,
  - IWDG – nastavení nezávislého watchdogu,
  - NVIC - nastavení systému přerušení,
  - RCC – nastavení reset signálu a distribuce systémového časování,
  - SYS – řízení režimu spánku,
  - WWDG – nastavení interního watchdogu.
- Analog – sekce pro nastavení analogových periferií
  - ADCx – nastavení analogově digitálních převodníků 1-3,
  - DAC - nastavení digitálně analogového převodníku.
- Timer – sekce na nastavení časovačů
  - LPTIM1 – nastavení časovače s nízkou spotřebou,
  - RTC – nastavení jednotky reálného času,
  - TIMx – nastavení obecných časovačů 1-14.
- Connectivity – nastavení komunikačních rozhraní
  - CANx – nastavení CAN sběrnicí 1-3,
  - ETH – nastavení ethernet rozhraní,
  - I2Cx – nastavení I^2C rozhraní 1-4,
  - SPIx – nastavení SPI sběrnic 1-6,
  - UARTx/USARTx – nastavení komunikace pomocí sériové linky.
- Multimedia – nastavení HDMI, I^2S, JPEG.
- Security – nastavení generátoru náhodných čísel.
- Computing – nastavení CRC, digitálních filtrů a Sigma-Delta modulátorů.
- Middleware – nastavení rozšiřujících knihoven FATFS, FreeRTOS, LwIP, LibJPEG.



Obr. 11 - Pracovního prostředí STM32CubeMX.

V pracovním prostředí (Obr. 11) je v pravé části reálné schéma rozložení nožiček procesoru STM32F767ZITx, kde žlutě jsou zbarveny nožičky, které jsou připravené pro využití, ale funkcionalita nabyla aktivována. Zeleně jsou zbarveny nožičky, jejichž funkcionalita zapnuta byla.

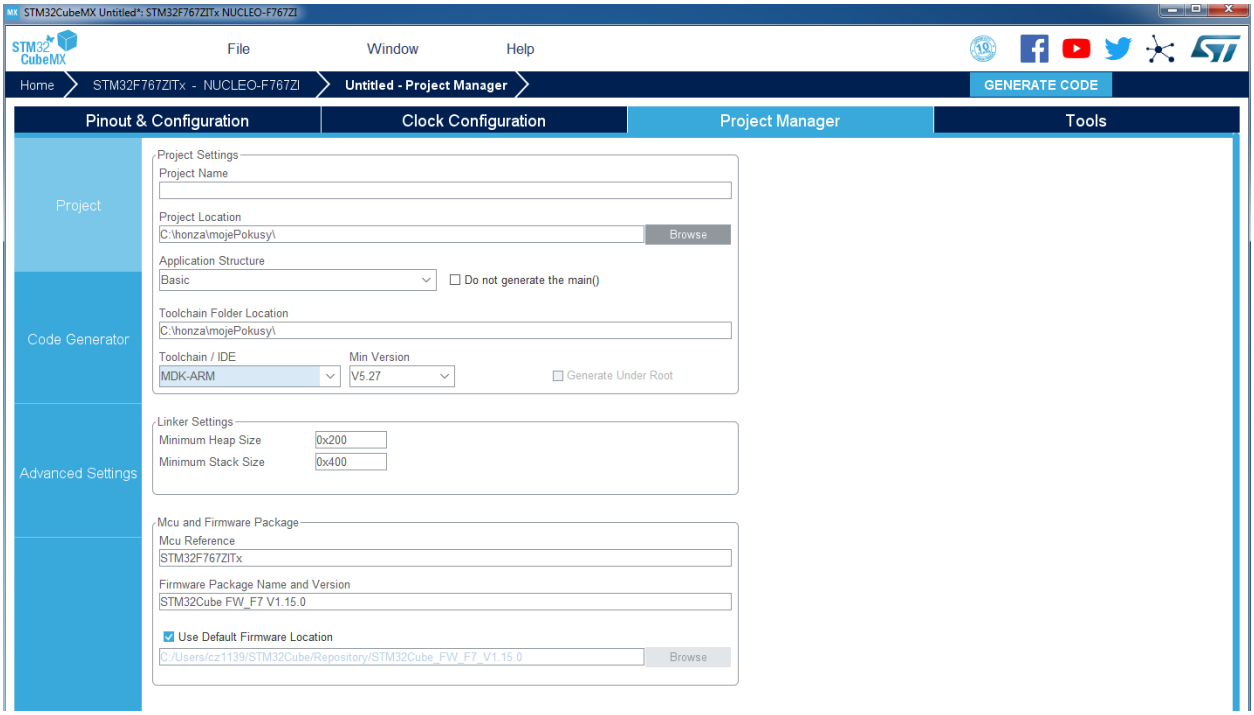
Prostřední část sekce „Pinout & Configuration“ je užívána k vlastnímu nastavení parametrů jednotlivých periférii.

5.2 Generování kostry projektu

Posledním krokem na celém projektu je generování jeho kostry, ale i přesto bude lepší jej zmínit už na začátku, aby bylo možné se na něj dále odkazovat.

V hlavní nabídce se provede volba sekce „Project Manager“ a vyplní se položka „Project Name“. Ta je nejen názvem projektu, ale také názvem podadresáře v adresáři „Project Location“.

Následně bude zvolen Toolchain/IDE, pro který se bude projekt generovat. V mém případě se jedná o „MDK-ARM“ a Min Version V5.27.



Obr. 12 - Nastavení parametrů pro generování kostry projektu.

Projekt bude následně vygenerován zmáčknutím tlačítka „GENERATE CODE“. Po vygenerování projektu se zobrazí dialogové okno, které nabídne možnost otevřít projekt ve zvoleném vývojovém prostředí.

5.3 Doporučení pro úpravu kódu

Vlastní program bude stačit upravovat v souboru „Src\main.c“, ve kterém jsou vloženy všechny podstatné funkce. Pro případné další rozšiřování projektu v STM32CudeMX je vhodné kód vpisovat mezi následující komentáře:

```
/* USER CODE BEGIN x */
/* USER CODE END x */
```

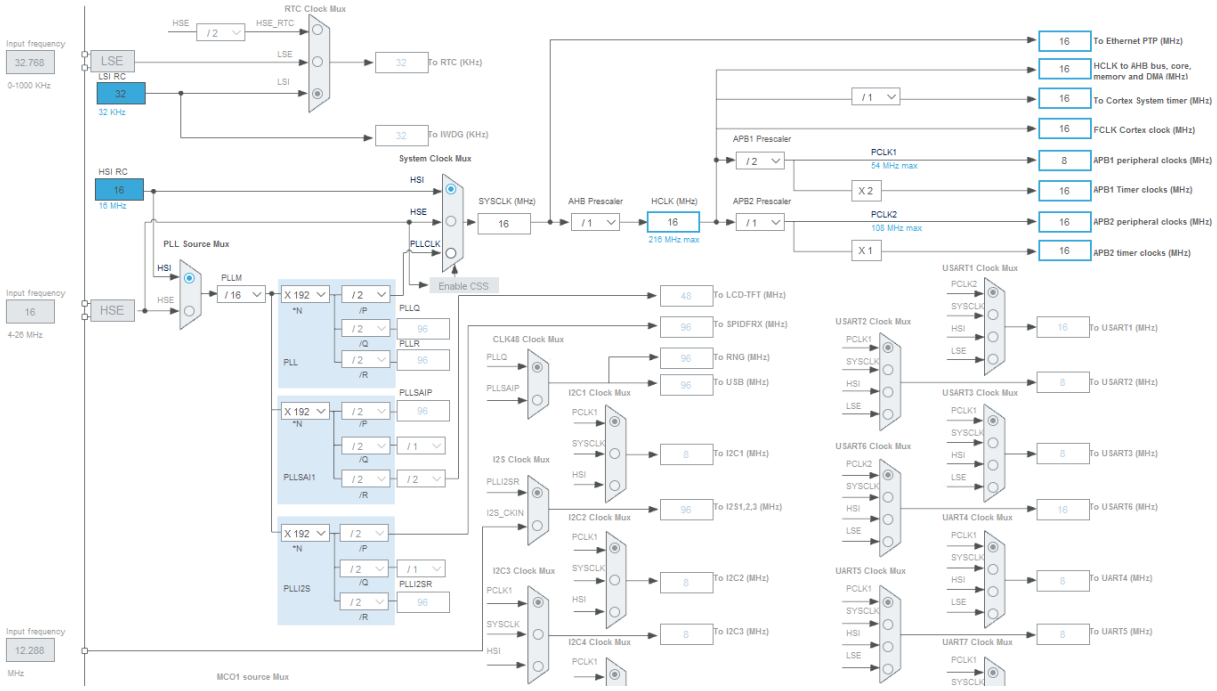
Při novém vygenerování projektu dojde k přepsání veškerého textu mimo tyto komentářové bloky.

5.4 Nastavení časování

Vývojová deska Nucleo-F767ZI je osazena mikroprocesorem STM32F767ZI, který umožňuje buď interní nebo externí zdroj signálu systémových hodin. Interním zdrojem je 16MHz HSI RC oscilátor. Externím oscilátorem může být použit např. 4-26 MHz. Z oscilátorů je signál přiveden

na fázový závěs, který z něj vytvoří signál systémových hodin o frekvenci až 216 MHz a tím je udáván takt pro všechny periferie. Sekce „Clock Configuration“ programu STM32CubeMX je zásadní pro nastavení všech časování – viz Obr. 13.

Frekvence jádra se nastavuje parametrem HCLK (MHz) –n a Obr. 13, je to modrý obdélník uprostřed. Po parametru HCLK jsou programem STM32CubeMX přepočítá všechny navázané časování. Část diagramu časování je zobrazena na Obr. 13.

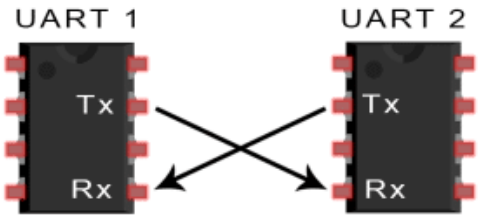


Obr. 13 - Diagram nastavení časovaní mikroprocesoru.

Návaznost časování na jednotlivé periferie bude vysvětlena u jednotlivých příkladů.

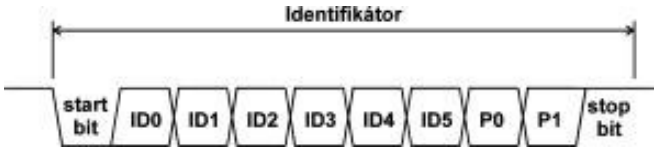
5.5 Obsluha UART rozhraní

Komunikace přes UART rozhraní je založena na sériové komunikaci, kdy je informace přenášena po komunikačním kanále pomocí dvou stavů L (LOW - nízký stav – nejčastěji 0 V) a H (HIGH - vysoký stav – nejčastěji 5 V). Komunikace se může účastnit jeden vysílač a jeden přijímač. Teoreticky by se dala sestavit komunikace jeden vysílač a více přijímačů (broadcast informace), ale v praxi se pro UART nepoužívá. Obousměrné spojení probíhá po dvou komunikačních kanálech Rx (receive – přijímací kanál) a Tx (transmit – vysílací kanál), Obr. 14.

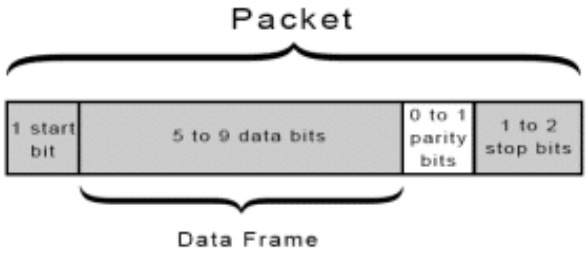


Obr. 14 - Zapojení zařízení komunikujících přes UART rozhraní [Circuit Basics, 2019].

Komunikační paket (Obr. 15 a Obr. 16) je sestaven z 1 start bitu, 5-9 datových bitů, 0-1 paritních bitů a 1-2 stop bitů.



Obr. 15 - Ukázka sériové komunikace [SUTORÝ,2019].



Obr. 16 - Paket sériové komunikace [Circuit Basics, 2019].

Start bit informuje příjemce o tom, že vysílač začal posílat data. Následuje sekvence datových bitů, které nesou přenášený datový balíček od vysílače k přijímači. Paritní bit je kontrolní součet sestavený z datových bitů a slouží jako kontrolní informace o tom, že data byla přenesena v pořádku. Stop bit nebo stop bity informují přijímač o tom, že přenos je ukončen.

Komunikace probíhá od vysílače k přijímači pouze po jednom jednosměrném komunikačním kanále. Je třeba zajistit mechanizmus, aby obě komunikující strany předaly informaci korektně. K tomu slouží pevná přenosová rychlost, která musí být před započetím přenosu nastavena na obou účastnících.

Přenosová rychlost se udává v jednotce Baud, která vypovídá o počtu možných změn na daném komunikačním kanále za sekundu.

Nejčastější zápis nastavení je ve formátu:

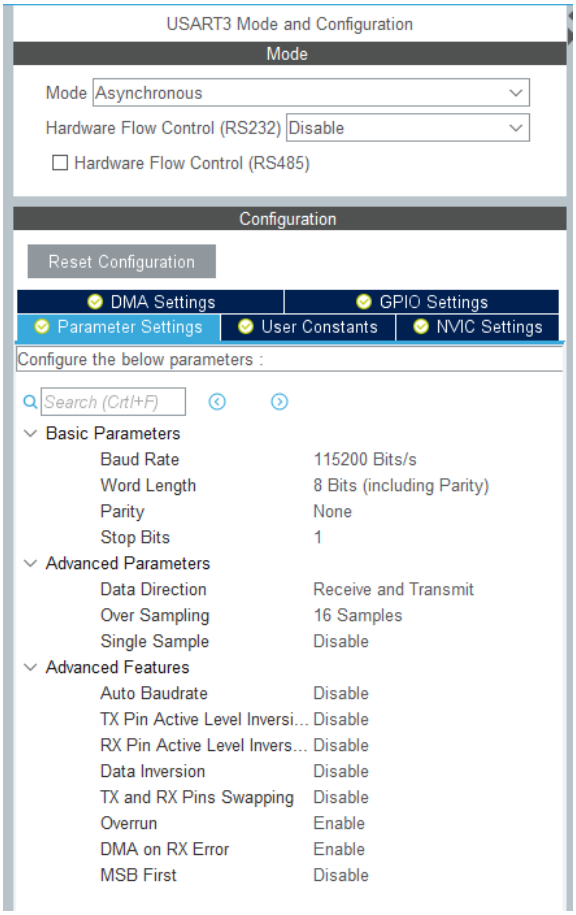
<rychlost> <počet datových bitů><parita><počet stop bitů>

- rychlost [baud] – nejčastěji 9600, 19200, 57600, 115200,
- počet datových bitů – 5 až 9,
- parita – N (none - žádná) , O (odd – lichý počet jedniček), E (even – sudý počet jedniček), M (mark – hodnota 1), S (space – hodnota 0),
- počet stop bitů – 1 nebo 2.

Například komunikace nastavená formátem **9600 8N1** označuje rychlost 9600 baudů, 8 datových bitů, žádná parita, jeden stop bit.

5.5.1 Realizace Tx UART komunikace

Kapitola bude věnována vlastní realizaci odeslání textového řetězce pomoci UART. Tato úloha je vhodná jednak pro odesílání informací o běhu programu a taktéž pro posílání povelů pro jiná zařízení ovládaná přes sériovou linku.



Obr. 17 - Přehled parametrů nastavení UART.

Nejprve je nutné připravit projekt v programu STM32CubeMX podle kroků v kapitole 5.1. Aktivaci UART rozhraní se provede v programu STM32CubeMX následovně:

- vybrat záložku „Pinout & Confifuration“,
- rozbalit v levém menu sekci „Connectivity“,
- zvolit UART4-8 nebo USART1,2,3,6 – v příkladu byl použit USART3,
- volbu Mode, v prostřední sekci Mode, přepnout na „Asynchronous“,
- v záložce „Parameter Settings“, v sekci „Configuration“ je třeba nastavit parametry Baud Rate, Word Lenght, Parity a Stop bit.

Postupnými kroky je vše připraveno a k vygenerování projektu. Tato akce je popsána v kapitole 5.2. Po vygenerování projektu se přechází do vývojového prostředí.

V souboru Src\main.c je nadefinována funkce `main()` předem připravené kostry programu pro komunikaci přes USART3. Funkce MX_USART3_UART_Init() nastavuje všechny parametry pro USART3, které jsou zvoleny v STM32CubeMX. Tato funkce je volána z funkce `main()`.

Pro odeslání řetězce se používá funkci HAL API s názvem HAL_UART_Transmit().

Funkce `HAL_UART_Transmit()` je definována takto:

```
HAL_StatusTypeDef HAL_UART_Transmit(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout)
```

Funkce vrací hodnoty `HAL_OK`, `HAL_ERROR`, `HAL_BUSY` nebo `HAL_TIMEOUT`. Pro případ kontroly správného provedení, je třeba kontrolovat pouze stav `HAL_OK`.

Parametrem `huart` je předán ovladač nadefinovaný programem STM32CubeMX, zde konkrétně huart3. V parametru `pData` je předán ukazatel na řetězec k odeslání, parametr `Size` bude obsahovat délku řetězce `pData` a parametr `Timeout` bude obsahovat, čas, jak dlouho bude trvat proces čekání, než bude pokus o odeslání řetězce prohlášený za nezdařený.

Celý kód vepsaný do předpřipraveného programu může vypadat takto:

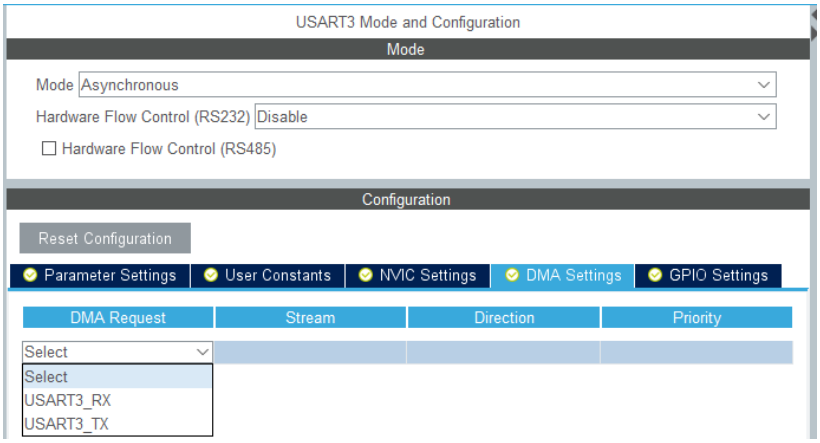
```
/* USER CODE BEGIN Includes */
// připojení standardní knihovny pro použití fce strlen()
#include <string.h>
/* USER CODE END Includes */

  /* USER CODE BEGIN 1 */
  // deklarace textu, který bud vypisován pomocí sériové linky
  char txData[40] = "Hello from STM32F7\r\n";
  /* USER CODE END 1 */
    /* USER CODE BEGIN 3 */
      //odeslání textu pomocí HAL funkce
      HAL_UART_Transmit(&huart5, (uint8_t *) txData, strlen(txData), 10);
      //HAL funkce na počkání zadaného času v milisekundách
      HAL_Delay(500);
  }
  /* USER CODE END 3 */
```

5.5.2 Realizace Rx UART komunikace pomocí DMA

Před analýzou nastavení komunikace nejprve vysvětlení principu komunikace pomocí DMA. Zkratka znamená v překladu přímý zápis do paměti. Jedná se o mechanizmus, který zjednoduší práci a také odlehčí procesoru při přesouvání dat mezi periferii a pamětí. Jedná se o periferie, které nakládají s různými daty, jako např. AD převodníky, UART, SD karty a tak podobně.

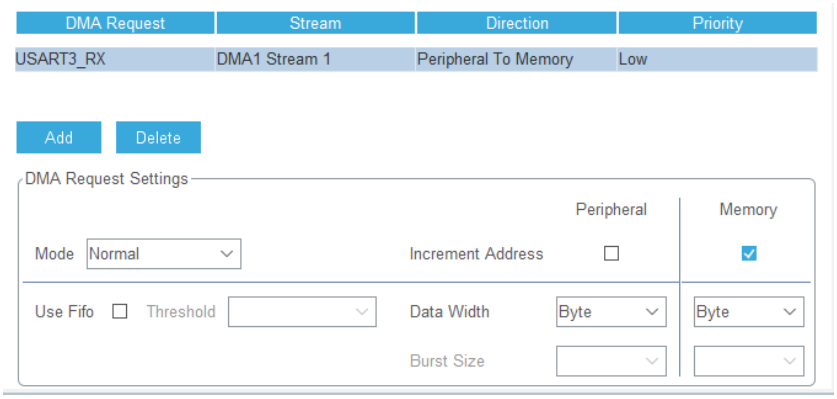
Aktivaci UART rozhraní s využitím DMA pro příjem dat, se provede v programu STM32CubeMX stejně jako v kapitole 5.5.1. Přidá se navíc nastavení DMA:

- V záložce DMA, v nastavení UART, pomocí tlačítka Add, je aktivován DMA kanál. Na výběr jsou možnosti Rx a Tx. DMA bude používáno pouze pro příjem, takže bude zvolena pouze možnost s Rx v názvu – viz Obr. 18



Obr. 18 - Přidání DMA kanálu pro UART.

- DMA kanál bude použit v módu Normal, kterým je zvyšována paměťová adresa proměnné, do které jsou zapisována data. Data budou datového typu Byte (Obr. 19).



Obr. 19 - Nastavení DMA pro příjem dat přes UART.

Těmito kroky je vše připraveno ke generování projektu. Tato akce je popsána v kapitole 5.2. Inicializace UART je provedena ve funkci `MX_USART3_UART_Init()` a inicializace DMA je provedena ve funkci `MX_DMA_Init()`. Tyto funkce vygeneroval program STM32CubeMX a jsou v nich nastavené všechny potřebné parametry.

Aktivaci Rx UART DMA se provede funkcí `HAL_UART_Receive_DMA()`. Její definice vypadá takto:

```
HAL_StatusTypeDef HAL_UART_Receive_DMA(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size)
```

Funkce vrací hodnoty `HAL_OK`, `HAL_ERROR`, `HAL_BUSY` nebo `HAL_TIMEOUT`. Pro případ kontroly správného provedení, bude potřeba kontrolovat pouze stav `HAL_OK`.

Parametr `huart` předá ovladač nadefinovaný od STM32CubeMX, ten je v mém případě `huart3`. V parametru `pData` se přenáší ukazatel na řetězec, do něhož se ukládají data z DMA. Parametr `Size` obsahuje délku přijímaného řetězce `pData`, při jejímž dosážení se vyvolá přerušení.

Kostra funkce, která je volána při přerušení je nadefinována v souboru Drivers\STMF32F7xx_HAL_Driver\stm32f7xx_hal_uart.c a je definována takto:

```
__weak void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
```

Do souboru „main.c“ je třeba překopírovat celou definici funkce, kromě direktivy `__weak`. Tato funkce se vyvolá pokaždé, když je obdržen počet znaků nastavených parametrem `Size` inicializační funkce. V této funkci je definován kód zpracovávající data přijatá přes sériovou linku.

Příklad obslužné rutiny pro příjem dat z UART pomocí DMA:

```
/* USER CODE BEGIN 0 */
// datový buffer pro příjem dat z UART pomocí DMA
char rxData[30];
/* USER CODE END 0 */
  /* USER CODE BEGIN 2 */
      HAL_UART_Receive_DMA(&huart5, (uint8_t *)rxData, 10);
  /* USER CODE END 2 */
```

```
/* USER CODE BEGIN 4 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
      // prostor pro kód zpracovávající data přijatá přes sériovou linku
}
/* USER CODE END 4 */
```

5.5.3 Zaznamenávání diagnostických informací přes virtuální sériový port

Pro lepší představu a ladění programu je dobré mít možnost sledovat, alespoň rámcově, zda se program nachází ve stavu, ve kterém je očekáván. Překladač odhalí řadu syntaktických chyb, ale bohužel je zatím krátký na sémantické. Tyto chyby dají odhalit až ve chvíli, kdy program dává jiný výstup, než je očekáváno nebo ovládané zařízení se chová jinak, než je očekáváno.

Vývojová deska Nucleo-F767ZI je vybavena ladícím modulem ST-LINK-V2, kterým je mikrokontrolér spojený s počítačem i sériovou linkou, která umožňuje posílat diagnostické informace. Tato sériová linka jen připojena na USART3.

Aktivaci sériové linky se provádí podle návodu v kapitole 5.5.1. Následně stačí upravit vygenerovanou kostru programu.

Odesílání diagnostických informací se provádí pomocí funkce `fprint`, která se předefinuje a to pomocí následujícího kódu:

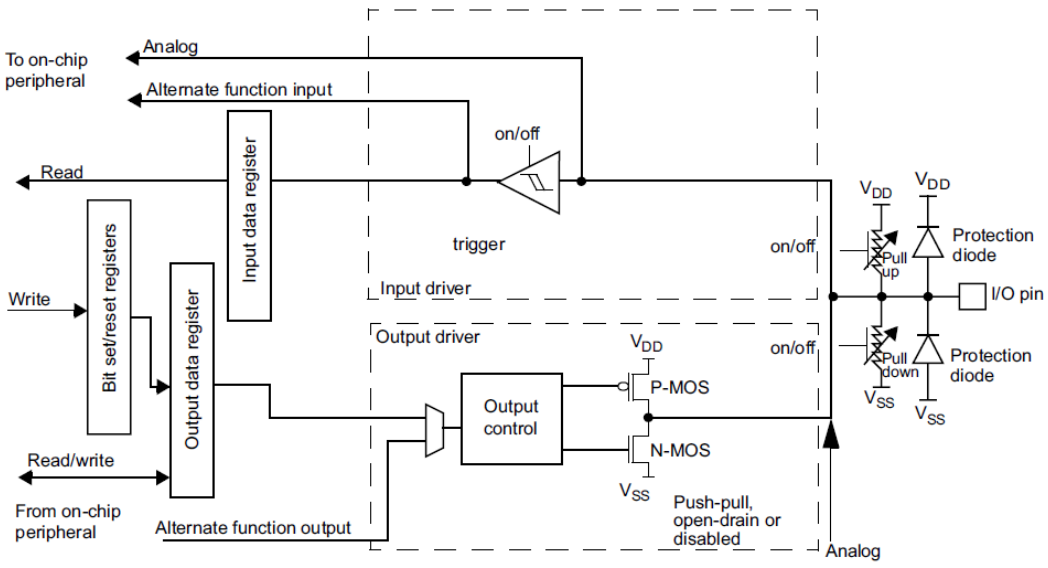
```
/* USER CODE BEGIN Includes */
// použijeme hlavičkový soubor s deklarací printf funkce
#include <stdio.h>
/* USER CODE END Includes */
/* USER CODE BEGIN PTD */
#ifdef __GNUC__
  /* With GCC, small printf (option LD Linker->Libraries->Small printf
     set to 'Yes') calls __io_putchar() */
  #define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
  #define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif /* __GNUC__ */
/* USER CODE END PTD */
```

```
/* USER CODE BEGIN 4 */
PUTCHAR_PROTOTYPE
{
  /* odeslání řetězce pomocí sériové linky */
  HAL_UART_Transmit(&huart3, (uint8_t *)&ch, 1, 0xFFFF);

  return ch;
}
/* USER CODE END 4 */
```

5.6 Ovládání digitálních vstupů a výstupů

Vstupy a výstupy mikrokontrolérů STM32 mají obecné využití, které se definuje pomocí jejich parametrů. Je možné je využít k připojení integrovaného ADC převodníku, integrovaného DAC převodníku, sériového rozhraní či jiných vnitřních periferií. Následující text bude zaměřen čistě na nízko úrovňovou komunikaci – změna výstup do logické úrovně 1 a 0.

Na Obr. 20 je vidět vnitřní zapojení obecných vstupů a výstupů mikrokontroléru STM32. Je zde patrné, že vstupy jsou chráněny proti přepětí i podpětí. Je možné je softwarově přepnout do nastavení s „pull-up“ nebo „pull-down“ rezistory nebo nechat výstup jako otevřený kolektor.



Obr. 20 - Vnitřní zapojení GPIO mikrokontroléru [STMicroelectronics, 2019].

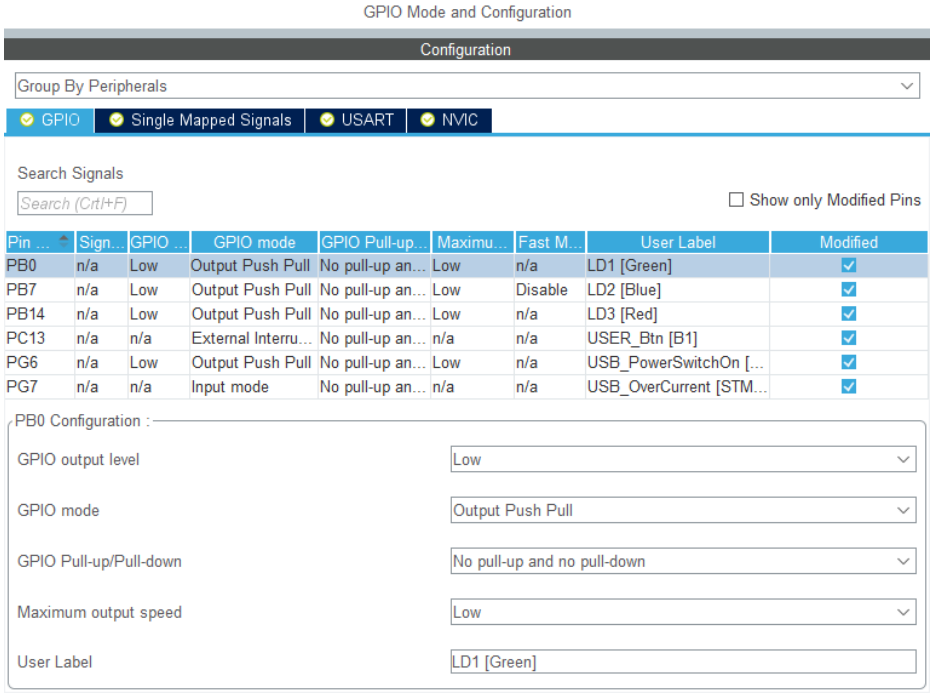
Následující popis je zaměřen na nastavení nebo čtení stavu z digitálních vstupů nebo výstupů. K ovládání výstupu je možné použít jednu ze tří uživatelských LED diod:

- zelená LED (LD1) – připojena na GPIO PB0,
- modrá LED (LD2) – připojena na GPIO PB7,
- červená LED (LD3) – připojena na GPIO PB14.

Pro vstupy bude použito uživatelské tlačítko:

- modré tlačítko (B1) – připojeno na GPIO PC13.

Použije se výchozí hodnota přednastavená pomocí programu STM32CubeMX (Obr. 21).



Obr. 21 - Výchozí nastavení přednastavených GPIO.)

5.6.1 Zjištění stavu GPIO

Čtení digitálního vstupu je prováděno pomocí HAL API funkce `HAL_GPIO_ReadPin()`. Ta je definována takto:

```
GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
```

Funkce vrací dvě hodnoty `GPIO_PIN_RESET` pro logickou úroveň 0 a `GPIO_PIN_SET` pro logickou úroveň 1.

Parametrem `GPIOx` je zvolen, jaký blok GPIO bude nastavován. K dispozici jsou GPIOA – GPIOK. Záleží na použitém mikrokontroléru, zda jsou k dispozici všechny.

Parematrem `GPIO_Pin` je nastaven, která z nožek mikrokontroléru v GPIO bloku bude nastavena. K dispozici jsou možnosti GPIO_PIN_0-15 a GPIO_PIN_ALL.

5.6.2 Nastavení stavu GPIO

Měna a nastaví stavu GPIO je možno dvěma funkcemi. HAL API `HAL_GPIO_TogglePin()`, která změní stav výstupu a `HAL_GPIO_WritePin()`, která nastaví stav na zvolený.

Definice funkcí je následující:

```
void HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)
void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
```

Parametrem GPIOx se volí, jaký blok GPIO budeme nastavován. K dispozici jsou GPIOA – GPIOK. Záleží na použitém mikrokontroléru.

5.6.3 Příklad čtení a nastavení GPIO

Nyní si následuje kód, jak přečíst a nastavit GPIO. Vstupní stavy GPIO se budou určeny pomocí stisknutí tlačítka B1 a změny výstupu budou patrné na rozsvícení LED, která bude kopírovat stisky tlačítka:

```
    /* USER CODE BEGIN 3 */
      if(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == GPIO_PIN_RESET)
      {
      //HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_14);
      HAL_GPIO_WritePin(GPIOB,GPIO_PIN_14,GPIO_PIN_SET);
      }
      else {
      HAL_GPIO_WritePin(GPIOB,GPIO_PIN_14,GPIO_PIN_RESET);
      }
      HAL_Delay(20);
  }
  /* USER CODE END 3 */
```

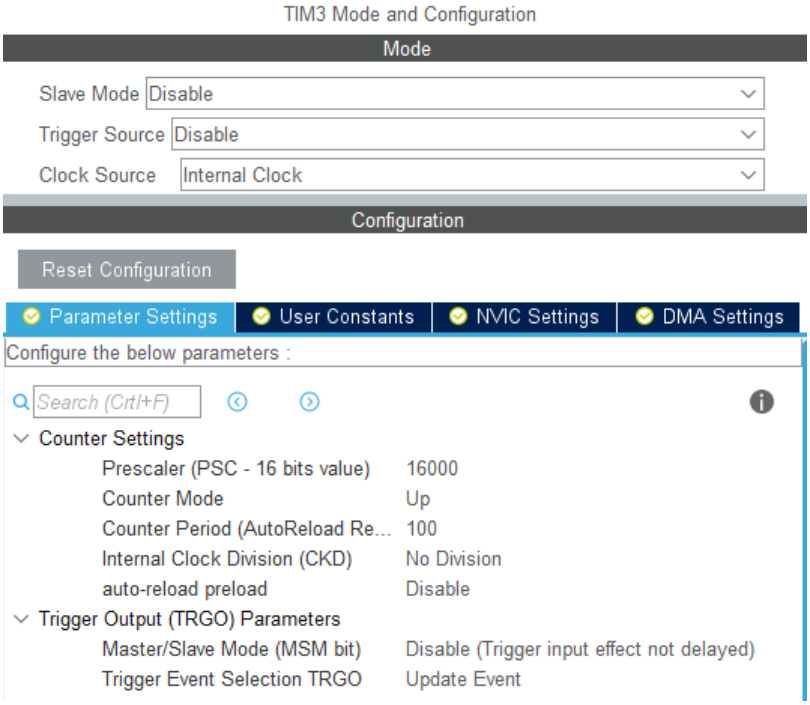
5.7 Časovač

Jednou z úloh, bez které se v oblasti řízení a automatizace nedá obejít, je měření času, vykonávání určité činnosti s pevnou periodou či měření frekvence. K tomu lze využít nástroj zvaný časovač.

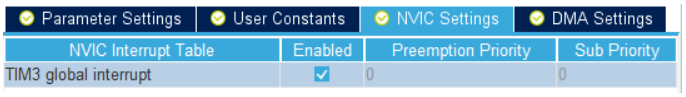
K dispozici je více než 10 časovačů. Nejčastěji použitá aplikace je vyvolání přerušení časovače po uplynutí stanovené doby a následně v obsluze přerušení vykonání potřebné akce.

Nastavení časovače v STM32CubeMX je implementováno v následujících krocích.

- Vybrán bude jeden časovač – např. TIM3.
- Aktivace se provede nastavením parametru „Clock source“ na „Internal Clock“.
- Parametr „Parametr Settings“ ->„Prescaler“ je 16bitová hodnota, která udává, jak bude vydělen zdroj časového signálu.
- Opakované spouštění časovače je zajištěno nastavením parametru „Parametr Settings“ -> „Trigger Event Slection TRGO“ na „Update Event“ – viz Obr. 22
- Vyvolání přerušení, po uplynutí času, zajistí zvolení „NVIC Settings“ na stav Enable – viz Obr. 23



Obr. 22 - nastavení parametrů časovače v STM32CubeMX



Obr. 23 - Aktivace přerušení k časovači.

Výpočet parametrů „Prescaler“ a „Counter period“:

- Každý časovač může být připojený na jinou časovou sběrnici, a proto je třeba zjistit na, kterou je připojený časovač TIM3. Tato informaci je k nalezení v Referenčním manuálu k mikroprocesoru [STMicroelectronics, 2019] - v mém případě se jedná o APB1.
- Následně je třeba ověřit nastavení „APB1 timer clocks“ (APB1TC) frekvence v sekci „Clock Configuration“ - v mém případě nastavena na 16MHz.

Peripheral	Bus	Peripheral	Bus
UART8	APB1	MDIOS	APB2
UART7		DFSDM1	
DAC		DSI Host	
PWR		LCD-TFT	
HDMI-CEC		SAI2	
CAN2		SAI1	
CAN1		SPI6	
I2C4		SPI5	
I2C3		TIM11	
I2C2		TIM10	
I2C1		TIM9	
UART5		EXTI	
UART4		SYSCFG	
USART3		SPI4	
USART2		SPI1	
SPDIFRX		SDMMC1	
SPI3 / I2S3		ADC1 - ADC2 - ADC3	
SPI2 / I2S2		SDMMC2	
CAN3		USART6	
IWDG		USART1	
WWDG		TIM8	
RTC & BKP Registers		TIM1	
LPTIM1			
TIM14			
TIM13			
TIM12			
TIM7			
TIM6			
TIM5			
TIM4			
TIM3			
TIM2			

Obr. 24 - Mapování časovačů na APB sběrnice.

Výpočet periody opakování časovače:

$$f_T = \frac{\text{APB1TC}}{Prescaler} = \frac{16\text{MHz}}{16000} = 1KHz$$

$$T_T = \frac{1}{f_T} \cdot CounterPeriod = \frac{1}{1\text{kHz}} \cdot 100 = 100ms$$

Nyní je vše připravené a může se přejít k vygenerování kostry programu.

Časovač se zapíná pomocí funkce `HAL_TIM_Base_Start_IT(),` která je definována takto:

```
HAL_StatusTypeDef HAL_TIM_Base_Start_IT(TIM_HandleTypeDef *htim)
```

Návratová hodnota je HAL_OK pokud je vše v pořádku. Pomocí parametru `htim` se předá funkci adresa ovladače časovače, který byl předefinovaný programem STM32CubeMX. Ten je v mém případě `htim3`.

Při každém přerušení od časovače je volána funkce `HAL_TIM_PeriodElapsedCallback()`. Funkci je třeba definovat a vložit do ní kód, který má být při přerušení zavolán.
Ukázkový příklad:

```
  /* USER CODE BEGIN 2 */
      HAL_TIM_Base_Start_IT(&htim3);
  /* USER CODE END 2 */


/* USER CODE BEGIN 4 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
      HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_14);
}
/* USER CODE END 4 */
```

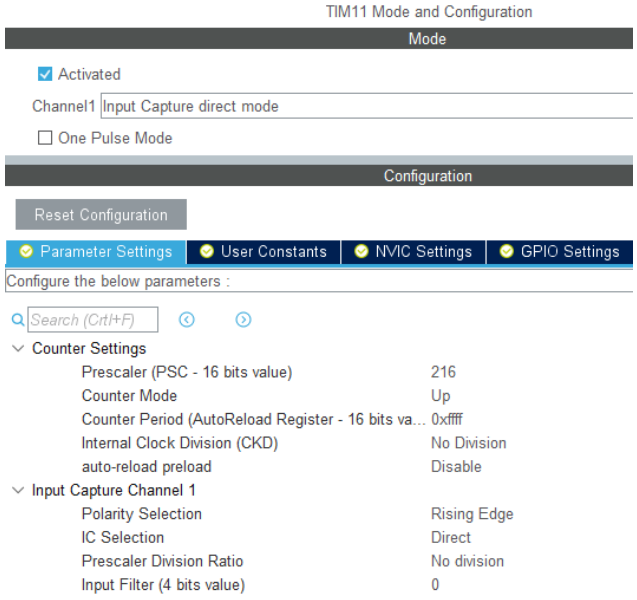
5.8 Úloha čítače s externím vstupem

Úloha využívá také časovač, ale v jiné funkci. Čítač počítá počet pulzů za určitý čas nebo signálu z externího zdroje.

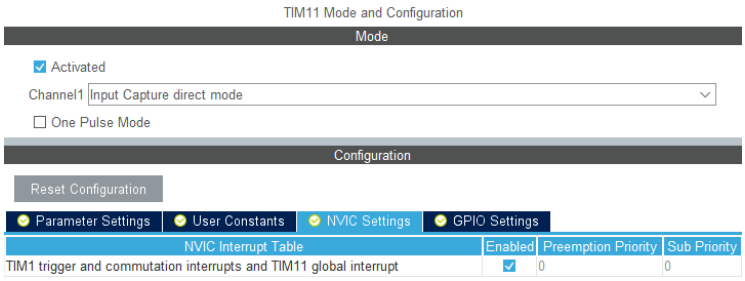
Nastavení časovače v STM32CubeMX se provede dle následujících kroků.

- Vybrán bude jeden časovač – např. TIM11.
- Aktivace se provede nastavením parametru „Clock source“ na „Input Capture direct mode“.
- Parametr „Parametr Settings“ ->„Prescaler“ je 16bitová hodnota, která udává, jak bude vydělen zdroj časového signálu.
- Parametr „Parametr Settings“ ->„Counter mode“ bude nastaven na „Up“.
- Parametr „Parametr Settings“ ->„Counter period“ bude nastaven na hodnotu „0xFFFF“.

- V sekci „Input Capture Channel 1“ bude parametr „Polarity Selection“ nastaven na „Rising Edge“. Tím bude čítač reagovat na vzestupnou hranu externího signálu.
- Zapnutí přerušení se provede dle Obr. 26.



Obr. 25 - Nastavení parametrů čítače v programu STM32CubeMX



Obr. 26 - Zapnutí přerušení pro čítač.

Čítač se zapíná funkcí HAL_TIM_IC_Start_IT() ta je definována takto:

```
HAL_StatusTypeDef HAL_TIM_IC_Start_IT(TIM_HandleTypeDef *htim, uint32_t Channel)
```

Návratová hodnota je `HAL_OK` pokud je vše v pořádku. Pomocí parametru `htim` se funkci předá adresa ovladače časovače, který byl předefinovaný programem STM32CubeMX. Ten je v mém případě `htim11`. Pomocí parametru „Channel“ je zvolen kanál časovače, který bude používán.

Vyčtení hodnoty časovače je provedeno pomocí funkce `HAL_TIM_ReadCapturedValue()`, která je definována takto:

```
uint32_t HAL_TIM_ReadCapturedValue(TIM_HandleTypeDef *htim, uint32_t Channel)
```

Návratovou hodnotou je počet pulzů časovače. Při každé náběžní nebo sestupné hraně pulzu z externího zdroje je volána funkce `HAL_TIM_IC_CaptureCallback()`, kterou je nutné znovu definovat a vložit do ní kód, který má být při přerušení volán.

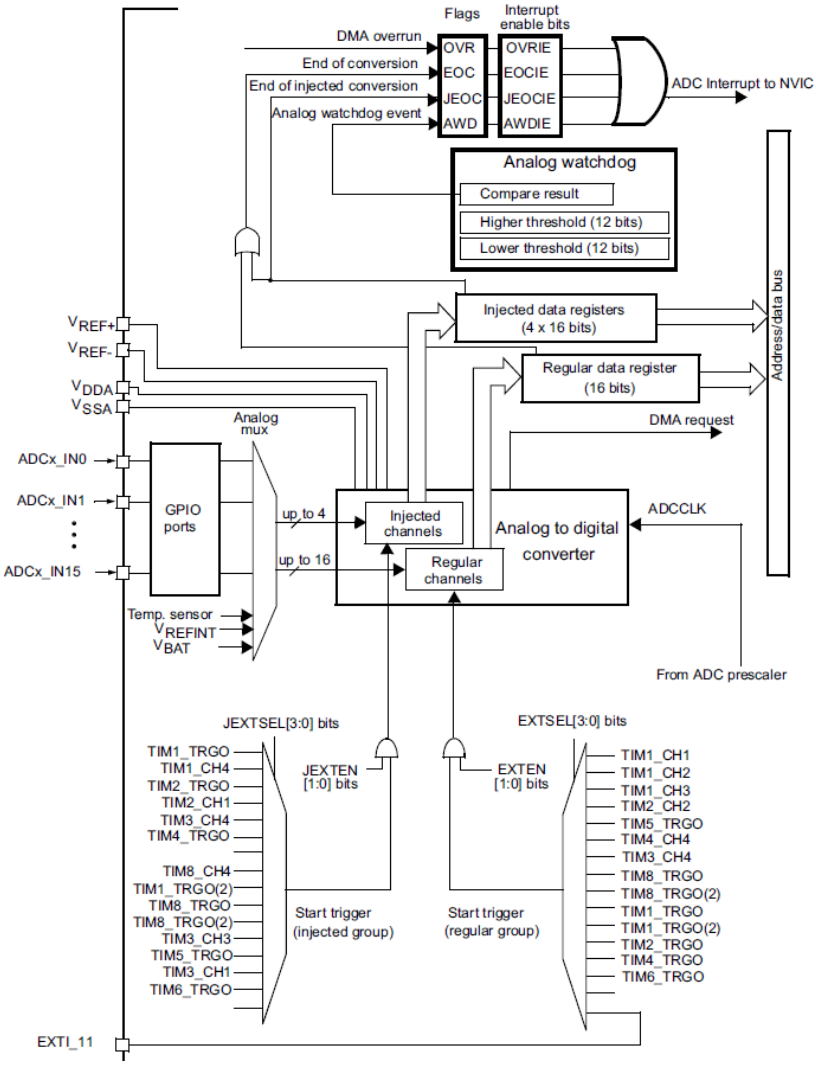
Ukázkový příklad:

```
  /* USER CODE BEGIN 2 */
      HAL_TIM_IC_Start_IT (&htim11);
  /* USER CODE END 2 */

/* USER CODE BEGIN 4 */
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
      HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_14);
}
/* USER CODE END 4 */
```

5.9 Úloha AD převodníku

Mikrokontroléry jsou digitální zařízení, které mají pouze dva stavy, tj. logický stav 1 nebo 0 (pravda nebo nepravda). Do těchto dvou stavů je potřeba převést celý okolní svět, který pracuje se spojitými signály, tj. že mezi každé dva změřené stavy lze vložit další hodnotu a toto štěpení lze provádět do nekonečna. Při převodu analogového (spojitého) signálu do prostředí digitálního (nespojitého) je třeba udělat ústupek a to v podobě kvantování a vzorkování analogového signálu. Tzn. stanovení vzorkovací periody, která se zvolí na základě Shanon-Kotělnikova teorému. A zajištění kvantování, tj. diskretizace oboru hodnot, nebo také převod spojitého signálu na konečný počet hodnot. Tento konečný počet hodnot je dán bitovou hloubkou (rozlišením) analogově-digitálního převodníku.

STM32F767ZI je vybaven třemi AD převodníky s postupnou aproximací, které lze nastavit na 12, 10, 8 nebo 6 bitové rozlišení. Každý převodník má 16 přepínaných vstupů.



Obr. 27 - Blokové signálové schéma AD převodníku v STM32.

5.9.1 Převod na vyžádání

Tato úloha vysvětluje, jak získat analogovou hodnotu z požadovaného vstupu AD převodníku ve zvolenou chvíli.

Nejdříve se připraví projekt v programu STM32CubeMX následujícím postupem.

- V sekci „Pinout & Configuration“, v sekci „Analog“ se zvolí jeden AD převodní – ADC1, ADC2 nebo ADC3 – volím ADC1.
- V sekci „Mode“ se vybere jeden vstup – volím IN13.
- Je zvoleno rozlišení převodníku na 12 bit parametrem „Resolution“.
- Volba času vzorkování parametrem „Sampling Time“ v sekci „ADC_Regular_ConversionMode“ -> Rank na 28 cyklů, což je více než potřebných 15.

Nyní je vše připraveno a může se provést generování projektu, viz kapitola 5.2. Pro zapnutí AD převodníku se použije funkce HAL API `HAL_ADC_Start()`, zapnutí konkrétního převodu se provádí pomocí funkce `HAL_ADC_PollForConversion()`, převedená hodnota je získána pomocí funkce `HAL_ADC_GetValue()` a vypnutí AD převodníku se provede funkcí `HAL_ADC_Stop()`.

Funkce jsou definovány takto:

```
HAL_StatusTypeDef HAL_ADC_Start(ADC_HandleTypeDef* hadc)

HAL_StatusTypeDef HAL_ADC_PollForConversion(ADC_HandleTypeDef* hadc, uint32_t
Timeout)

HAL_StatusTypeDef HAL_ADC_Stop(ADC_HandleTypeDef* hadc)

uint32_t HAL_ADC_GetValue(ADC_HandleTypeDef* hadc)
```

Každá funkce, pokud proběhne bez problému, vrátí hodnotu `HAL_OK`. Toto neplatí pro funkci `HAL_ADC_GetValue`, která vrací převedenou hodnotu.

Parametrem `hadc` je předán funkcím adresa handle do AD převodníku. Tuto proměnnou již nadefinoval program STM32CubeMX. V mém případě je to `hadc1`.

Parametr `Timeout` udává, po jakém čase bude konverze prohlášena za nezdařenou.

Ukázkový příklad:

```
/* USER CODE BEGIN 0 */
// definice proměnné pro uchování hodnoty z AD převodníku
uint32_t adcValue;
/* USER CODE END 0 */

    /* USER CODE BEGIN 3 */
      // zapnutí ADC převodníku s handle hadc1
      HAL_ADC_Start(&hadc1);
      // zapnutí konverze
      if(HAL_ADC_PollForConversion(&hadc1, 5) == HAL_OK)
      {
```

```
    // konverze proběhla úspěšně, vyčteme převedenou hodnotu
    adcValue = HAL_ADC_GetValue(&hadc1);
    }
    // zastavíme převodník a tím vymažeme všechny jeho registry a uvedeme
    // do původního stavu
    HAL_ADC_Stop(&hadc1);
    // počkáme 100ms
    HAL_Delay(100);
}
/* USER CODE END 3 */
```

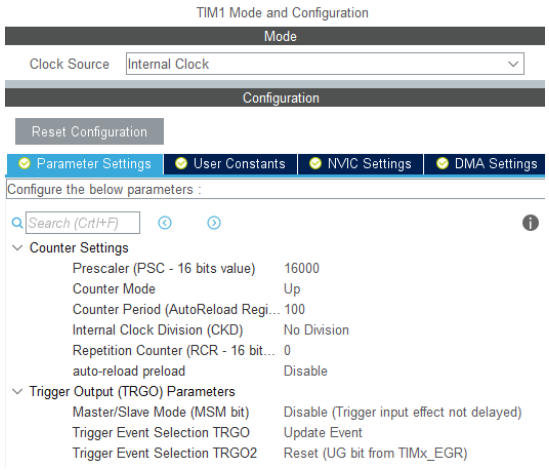
5.9.2 Periodický převod s danou periodou

Způsob, jak získat analogovou hodnotu z požadovaného vstupu AD převodníku v čase definovaném přerušením od časovače, je popsán v následujících řádcích.

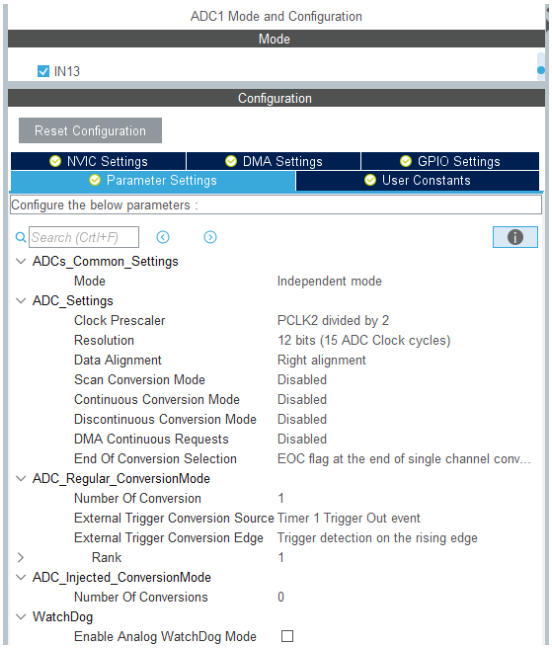
Nejdříve je třeba připravit projekt v programu STM32CubeMX podle daného postupu.

- Nastavení časovače TIM1 – viz Obr. 28:
    - Vzorkování bude nastaveno na 100ms.
    - Pro „APB2 timer clock“ 16MHz je zvolen „Prescaler“ na hodnotu 16000 a „Counter Period“ na hodnotu100.
    - Parametr „Trigger Output“ -> „Trigger Event Selection TRGO“ je nastaven na „Update Event“, ať se znovu zapne časování po prvním přerušení.
- V sekci „Pinout & Configuration“, v sekci „Analog“ je vybrána v položce ADC, jeden AD převodní – ADC1, ADC2 nebo ADC3 – volím ADC1.
- Následně je zvolen jeden vstup v sekci „Mode“ – volím IN13.
- Parametr „Resolution“ převodníku nastavíme na 12 bit.
- Parametr „Sampling Time“ v sekci „ADC_Regular_ConversionMode“ -> Rank je nastaven na 28 cyklů, což je více než potřebných 15.
- V sekci „ADC_Regular_ConversionMode“ je nastaven parametr „External Trigger Conversion Source“ na „Timer 1 Trigger Out event“ – tato volba zajistí, že při přerušení od Timer1 dojde ke spuštění konverze AD převodníku – viz Obr. 27

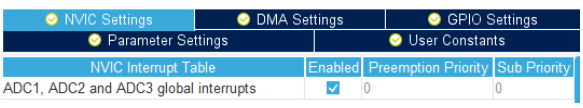
- V záložce „NVIC Settings“ je zašrktnuto „Enable“ – tímto se aktivuje přerušení od AD převodníku, viz Obr. 30.



Obr. 28 - Nastavení časovače pro generování periody vzorkování.



Obr. 29 - Nastavení AD převodníku pro spuštění přerušením Timer1.



Obr. 30 - Zapnutí přerušení AD převodníku po dokončení převodu.

Nyní je vše připraveno a může se nechat vygenerovat kostra projektu – viz kapitola 5.2.

Pro ovládání časovače budou využívány funkce HAL API. Pro zapnutí časovače funkce `HAL_TIM_Base_Start()`, zapnutí AD převodníku v módu přerušení `HAL_ADC_Start_IT()`, a získání převedené hodnoty pomocí funkce `HAL_ADC_GetValue()`.

Funkce jsou definovány takto:

```
HAL_StatusTypeDef HAL_TIM_Base_Start(TIM_HandleTypeDef *htim)

HAL_StatusTypeDef HAL_ADC_Start_IT(ADC_HandleTypeDef* hadc)

uint32_t HAL_ADC_GetValue(ADC_HandleTypeDef* hadc)
```

Každá funkce, pokud proběhne bez problému, má návratovou hodnotu `HAL_OK`. Toto neplatí pro funkci `HAL_ADC_GetValue`, která vrací převedenou hodnotu.

Parametr `htim` předá funkcím adresu ovladače časovače. Tuto proměnnou nadefinoval program STM32CubeMX. V mém případě je to `htim1`.

Parametrem `hadc` předá funkcím adresu handle AD převodníku. Tuto proměnnou nadefinoval program STM32CubeMX. V mém případě je to `hadc1`.

Při každém přerušení od AD převodníku je volána funkce `HAL_ADC_ConvCpltCallback()`, kterou je třeba předefinovat a vložit do ní kód volaný při přerušení.

Ukázkový příklad:

```
/* USER CODE BEGIN 0 */
// globální proměnná, do které je ukládána zkonvertovaná hodnota z přerušení
uint16_t adcVal;
/* USER CODE END 0 */

  /* USER CODE BEGIN 2 */
      // zapnutí časovače, který periodicky spouští AD převodník
      HAL_TIM_Base_Start(&htim1);
      // zapnutí AD převodníku v módu přerušení
      HAL_ADC_Start_IT(&hadc1);
  /* USER CODE END 2 */
```

```
/* USER CODE BEGIN 4 */
// předefinovanná funce přerušení AD převodníku
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
      // uložení zkonvertované hodnoty do globální proměnné
      adcVal = HAL_ADC_GetValue(&hadc1);
}
/* USER CODE END 4 */
```

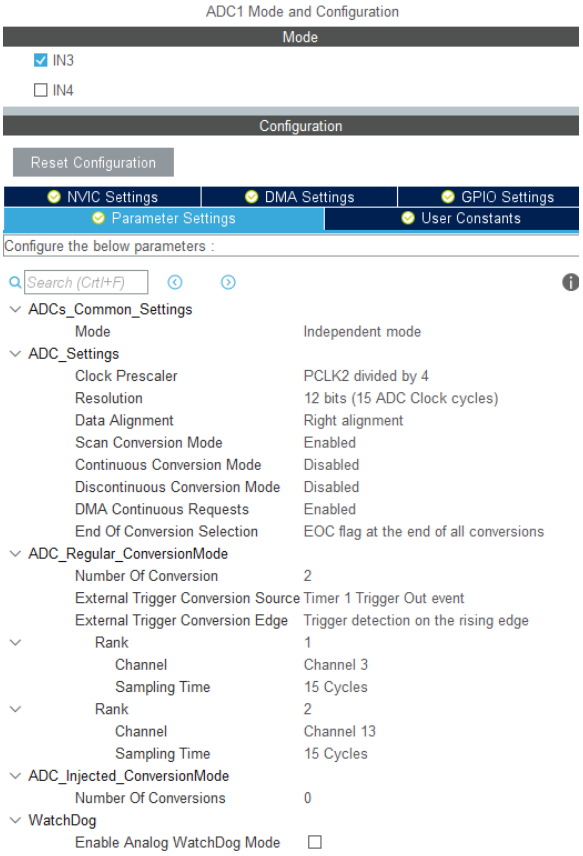
5.9.3 Periodický převod s danou periodou a použitím DMA s více vstupy

V této úloze je ukázáno, jak získat analogovou hodnotu z více požadovaných vstupů AD převodníku, v čase definovaném přerušením od převodníku. Data se budou následně uložena do paměti pomocí mechanizmu DMA.

Nejdříve je potřeba připravit projekt v programu STM32CubeMX:

- Nastavení časovače TIM1 – viz Obr. 31:
  - Vzorkování je nastaveno na 100 ms.
  - Při „APB2 timer clock“ 16 MHz je „Prescaler“ nastaven na 16000 a „Counter Period“ na 100.
  - Parametr „Trigger Output“ -> „Trigger Event Selection TRGO“ je nastaven na „Update Event“, ať se znovu zapne časování po prvním přerušení.
- V sekci „Pinout & Configuration“, v sekci „Analog“, je vybrán jeden AD převodník – ADC1, ADC2 nebo ADC3 – volím ADC1.
- Následně jsou zvoleny vstupy v sekci „Mode“ – volím IN3 a IN13.
- Parametr „Resolution“ převodníku je nastaven na 12 bit.
- Parametr „Scan Conversion Mode“ je nastaven na Enable, tzn. při každém měření se provede konverze všech zapnutých vstupů.
- Parametr „DMA Continues Request“ je nastaven na Enable. Tento parametr zajistí zapnutí DMA.
- V sekci „ADC_Regular_ConversionMode“ je třeba nastavit:
  - „Number of Conversion“ na 2 – jsou zapnuté 2 vstupy.

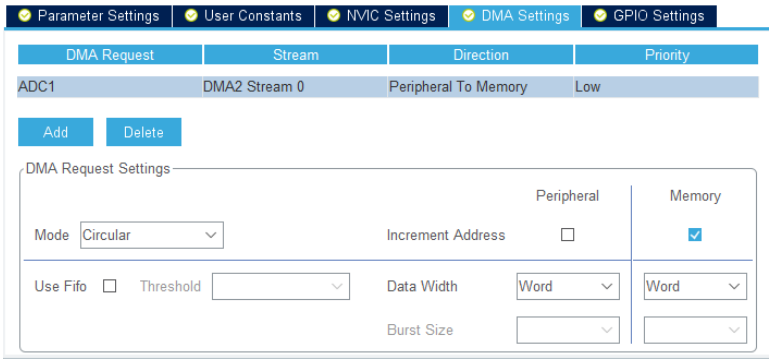
- V sekci Rank 1 „Channel“ na „Channel 3“ a „Sampling Time“ na 15 cyklů – nastavení 1. konverzního cyklu pro vstupní kanál 3.
- V sekci Rank 2 „Channel“ na „Channel 13“ a „Sampling Time“ na 15 cyklů – nastavení 2. konverzního cyklu pro vstupní kanál 13.
- „External Trigger Conversion Source“ na „Timer 1 Trigger Out event“ – tato volba zajistí, že při přerušení od Timer1 dojde ke spuštění konverze AD převodníku – viz Obr. 31.



Obr. 31 - Nastavení parametrů AD převodníku pro DMA přenos a spuštění časovačem.

- V záložce „NVIC Settings“ se zaškrtne „Enable“ – tímto je aktivováno přerušení od AD převodníku – viz Obr. 23.
- V záložce „DMA Settings“ se zvolí ADC1 a přenastaví následující parametry (viz Obr. 32):
  - „Mode“ na hodnotu Circular – zajisti, že při každém přenosu se ukazatel přesune na začátek proměnné pro ukládání dat.

- „Increment Address“ se zaškrtne pro část Memory.
- „Data Width“ se nastaví pro část Memory na hodnotu Word.



Obr. 32 - Nastavení DMA parametrů pro přenos dat z AD převodníku.

Nyní je vše připraveno a může se nechat vygenerovat kostra projektu – viz kapitola 5.2. Budou využívány funkce HAL API pro zapnutí časovače `HAL_TIM_Base_Start()` a zapnutí AD převodníku v módu DMA `HAL_ADC_Start_DMA()`.

Funkce jsou definovány takto:

```
HAL_StatusTypeDef HAL_TIM_Base_Start(TIM_HandleTypeDef *htim)
HAL_StatusTypeDef HAL_ADC_Start_DMA(ADC_HandleTypeDef* hadc, uint32_t* pData,
uint32_t Length)
```

Každá funkce, pokud proběhne bez problému, vrátí hodnotu `HAL_OK`. Parametrem `htim` předává funkcím adresu handle časovače. Tuto proměnnou nadefinoval program STM32CubeMX. V mém případě je to `htim1`. Parametrem `hadc` předává funkcím adresu handle AD převodníku. Tuto proměnnou nadefinoval program STM32CubeMX. V mém případě je to `hadc1`. Parametrem `pData` předá funkci pole o dvou prvcích, do kterého se bude DMA přenášet data od AD převodníku. Parametrem `Length` předává funkci informaci o počtu prvku v poli, konkrétně zde 2.

Při každém přerušení od AD převodníku je volána funkce `HAL_ADC_ConvCpltCallback()`, kterou je třeba předefinovat a vložit do ní kód, který má být vykonáván při přerušení.

Ukázkový příklad:

```
/* USER CODE BEGIN 0 */
uint32_t adcVal[2];
/* USER CODE END 0 */
```

```
  /* USER CODE BEGIN 2 */
      // zapnutí časovače, který periodicky spouští AD převodník
      HAL_TIM_Base_Start(&htim1);
      // zapnutí AD převodníku v módu DMA
      HAL_ADC_Start_DMA(&hadc1, adcVal, 2);
  /* USER CODE END 2 */

/* USER CODE BEGIN 4 */
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
      // kód pro další zpracování zkonvertovaných dat
}
/* USER CODE END 4 */
```

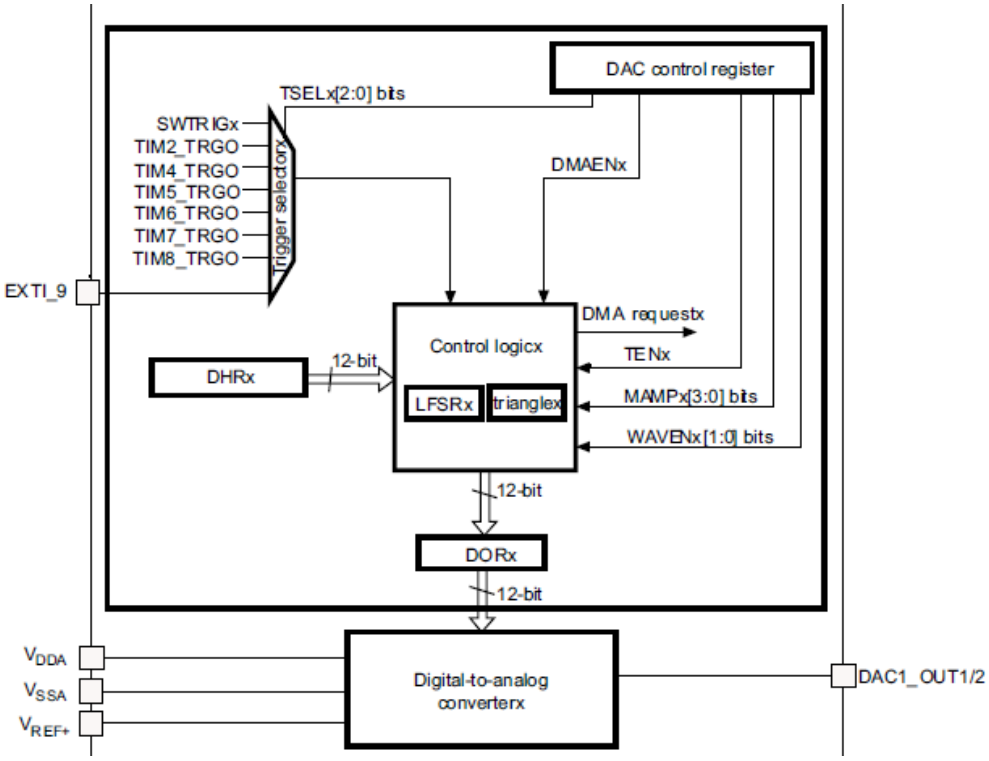
5.10 Úloha DA převodníku

Díky faktu, že celý náš svět je založený na dějích, jejichž popis je dán spojitými funkcemi, tak i při komunikaci mikrokontroléru s okolím je potřeba mít možnost sestrojit z diskrétního signálu signál spojitý nebo alespoň co nejvíce se blížící alternativu. K tomu je k dispozici DA převodník.

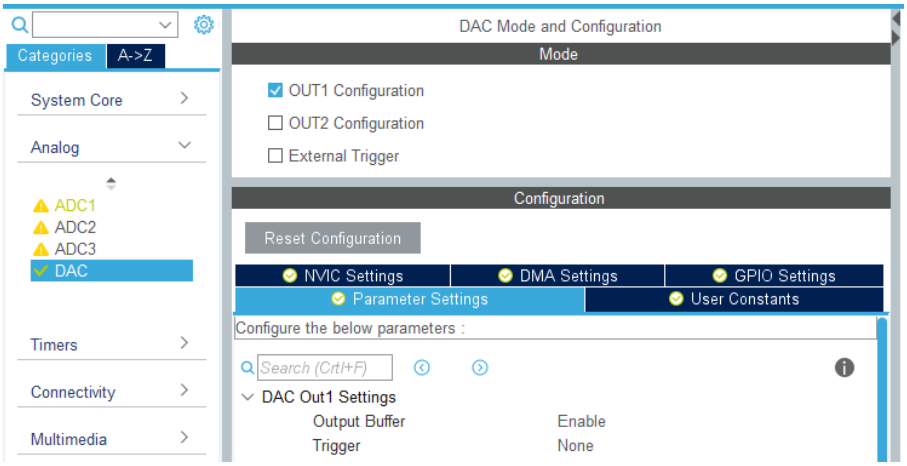
STM32F767ZI je vybaven jedním DA převodníkem, který lze nastavit na 12 nebo 8 bitové rozlišení. Převodník má dva výstupní kanály.

K dalším krokům je zapotřebí mít připravený projekt v programu STM32CubeMX podle následujících kroků.

- V sekci „Pinout & Configuration“, v sekci „Analog“ se zvolí položka DAC.
- V sekci „Mode“ se zaškrtne OUT1.



Obr. 33 - Blokové signálové schéma DA převodníku v STM32.



Obr. 34 - Nastavení DA převodníku v programu STM32CubeMX.

Nyní je vše připraveno a je možné vygenerovat projekt – viz kapitola 5.2.

Budou využívány funkce HAL API `HAL_DAC_Start()`, pro zapnutí DA převodníku a `HAL_DAC_SetValue()`, pro nastavení hodnoty, kterou bude DA převodník převádět.

Funkce jsou definovány takto:

```
HAL_StatusTypeDef HAL_DAC_Start(DAC_HandleTypeDef* hdac, uint32_t Channel)
HAL_StatusTypeDef HAL_DAC_SetValue(DAC_HandleTypeDef* hdac, uint32_t Channel,
uint32_t Alignment, uint32_t Data)
```

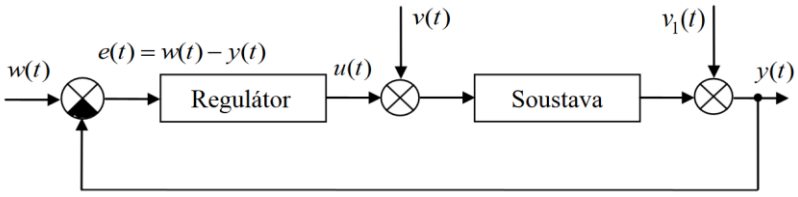
Každá funkce, pokud proběhne bez problému, vrátí hodnotu `HAL_OK`. Parametrem `hdac` předá funkcím adresu ovladače DA převodníku. Tuto proměnnou nadefinoval program STM32CubeMX. V mém případě je to `hdac`. Parametrem `Channel` je předáno funkcím, jaký kanál DA převodníku mají zvolit. V mém případě to je `DAC_CHANNEL_1`. Parametr `Alignment` předá funkcím, jaké zarovnání se DA převodník bude používat. V mém případě to je `DAC_ALIGN_12B_R`. Parametrem `Data` předá funkcím, hodnotu, kterou DA převodník převede na analogovou.

Ukázkový příklad:

```
     /* USER CODE BEGIN 2 */
    // hodnota pro převod DA převodníkem
    uint32_t dacVal = 0;
    // směr inkrementace
    uint32_t direction = 0;
    // zapnutí kanálu 1 DA převodníku
    HAL_DAC_Start(&hdac,DAC_CHANNEL_1);
    // nastavení nulové hodnoty DA převodníkem
    HAL_DAC_SetValue(&hdac,DAC_CHANNEL_1,DAC_ALIGN_12B_R, dacVal);
    /* USER CODE END 2 */

  /* USER CODE BEGIN 3 */
    // postupné zapisování hodnot a generování pilového signálu
    HAL_DAC_SetValue(&hdac,DAC_CHANNEL_1,DAC_ALIGN_12B_R,dacVal);
    if(direction == 0)dacVal ++;
    else dacVal --;

    if(dacVal == 0) dacVal = 1;
    if(dacVal == 4095) dacVal = 0;
}
/* USER CODE END 3 */
```

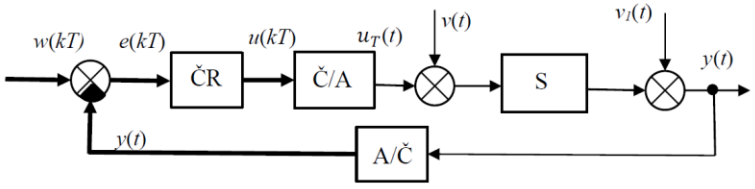
6 Proces regulace v oblasti mikrokontrolérů

Řízení probíhá v regulačním obvodu (Obr. 35). Oproti ovládání je to zpětnovazební proces, kdy na vstup regulátoru se přivádí regulační odchylka $e(t)$, získaná odečtením žádané hodnoty $w(t)$ a regulované veličiny $y(t)$. Cíl regulace je dán splněním podmínky $e(t) \rightarrow 0$ nebo $y(t) \rightarrow w(t)$.



Obr. 35 - Regulační obvod.

V oblasti mikrokontrolérů je do regulačního obvodu třeba zakomponovat fakt, že mikrokontrolér je digitální zařízení, které vystupuje v roli číslicového regulátoru (ČR) a je napojené na analogovou soustavu. Toho se dosáhne tím, že je do zpětné vazby vložen AD převodník a na vstup soustavy DA převodník (Obr. 36).



Obr. 36 - Číslicový regulační obvod.

V případě této diplomové práce v návaznosti na využití signálových procesorů pro regulaci laboratorního modelu aplikuji číslicovou regulaci.

6.1 Implementace PID regulátoru

V analogových obvodech je PID regulátor popsán rovnicí (1).

$$u(t) = k_p \left[e(t) + \frac{1}{T_I} \int_0^t e(\tau) d\tau + T_D \frac{de(t)}{dt} \right] \tag{1}$$

Pro implementaci rovnice (1) v mikrokontroléru je zapotřebí rovnici získat v číslicové podobě. Z tohoto důvodu se čas t nahradí disktrétním časem kT, dále se nahradí integrace sumací (2) a derivace zpětnou diferencí (3).

$$\int_0^t e(\tau)d\tau \;\rightarrow\; T\sum_{i=0}^{k} e(iT), \tag{2}$$

$$\frac{de(t)}{dt} \rightarrow \frac{\nabla e(kT)}{T}, \tag{3}$$

čímž je získána rovnice číslicového PID regulátoru neboli PSD regulátoru.

$$u(kT) = \; k_p\left[e(kT) + \frac{T}{T_I}\sum_{i=0}^{k} e(iT) + \frac{T_D}{T}\nabla e(kT)\right]. \tag{4}$$

Zápis (4) rovnice PSD regulátoru se nazývá polohový algoritmus. Pro implementaci PSD regulátoru v mikrokontrolérech se nejčastěji aplikuje přírůstkový tvar. Ten neobsahuje problematickou sumaci. V přírůstkovém algoritmu se nevypočítává přímo aktuální hodnota *u*(*kT*), ale pouze její přírůstek oproti hodnotě *u*[(*k-1*)*T*] vztahem

$$u[(k-1)T] = \; k_p\left[e[(k-1)T] + \frac{T}{T_I}\sum_{i=1}^{k-1} e(iT) + \frac{T_D}{T}\nabla e[(k-1)T]\right]. \tag{5}$$

Po odečtení výrazů (5) a (6) je získán přírůstkový tvar PSD regulátoru dle vztahu

$$\nabla u(kT) = \; k_p\left[\nabla e(kT) + \frac{T}{T_I}e(kT) + \frac{T_D}{T}\nabla^2 e(kT)\right], \tag{6}$$

kde

$$\nabla^2 e(kT) = \; \nabla e(kT) - \nabla e[(k-1)T] = e(kT) - 2e[(k-1)T] + \; e[(k-2)T]. \tag{7}$$

Po dosazení je získán vhodný tvar pro implementaci.

$$\begin{aligned} u(kT) - u[(k-1)T] \; = k_p\Big(e(kT) - e[(k-1)T] + \tfrac{T}{T_I}e(kT) + \\ \tfrac{T_D}{T}\{e(kT) - 2e[(k-1)T] + \; e[(k-2)T]\}\Big). \end{aligned} \tag{8}$$

Zavedeme substituci

$$q_0 \; = \; k_p\left(1 + \frac{T}{T_I} + \frac{T_D}{T}\right), \tag{9}$$

$$q_1 \; = \; -k_p\left(1 + 2\frac{T_D}{T}\right), \tag{10}$$

$$q_2 \; = \; k_p\frac{T_D}{T}, \tag{11}$$

čímž je získán tvar:

$$u(kT) = u[(k-1)T] + q_0 e(kT) + q_1 e[(k-1)T] + q_2 e[(k-2)T]. \quad (12)$$

Tuto rovnici jsem implementoval následujícím kódem v jazyce C pro STM32:

```
//PID stuktura, která v sobě nese veškeré informace o číslicovém PID regulátoru
typedef struct
{
	float q0;// diferencni koeficient, q0 = Kp + Ki + Kd
	float q1;// diferencni koeficient, q1 = -Kp - 2Kd
	float q2;// diferencni koeficient, q2 = Kd
	float w;// zadana velicina
	float e[3];// regulacni odchylka - e(kT), e[(k-1)T], e[(k-2)T]
	float u[2];// akcni velicina - u(kT), u[(k-1)T]
	float Kp;// proporcionalni koeficient
	float Ti;// integracni casova konstanta
	float Td;// derivacni casova konstanta
	float T;// perioda
	float limit_min;// min. omezeni vystupu
	float limit_max;// max. omezeni vystupu
	uint8_tI;// zapnuti I slozky
	uint8_tD;// zapnuti D slozky
} pid_instance;

// deklarace PID proměnné
pid_instance PID;
// makro na omezení hodnoty
#define constrain(amt,low,high) ((amt)<(low)? (low): ((amt)>(high) ? (high):
(amt) ) )

// funkce pro inicializaci parametrů q0, q1 a q2
void pid_init( pid_instance * S )
{
	S->q0 = 0;
	S->q1 = 0;
```

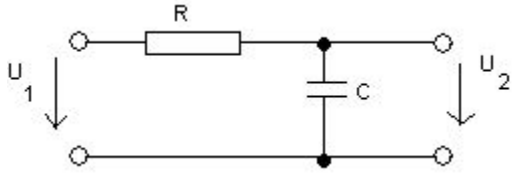
```
      S->q2 = 0;
      // propocet koeficientu pro proporcionalni slozku
      // koeficient q0
  S->q0 = S->Kp*1;
  // koeficient q1
  S->q1 = (-S->Kp)*1;

      if (S->I == 1){
      // pricteni koeficientu pro integracni slozku
      // koeficient q0
      S->q0 = S->q0 + (S->Kp*S->Ti/S->T);
      }
      if (S->D == 1){
      // pricteni koeficientu pro derivacni slozku
      // koeficient q0
      S->q0 = S->q0 + (S->Kp*S->T/S->Td);
      // koeficient q1
      S->q1 = S->q1 + (-S->Kp)*((float) 2.0 * S->Td/S->T);
      // koeficient q2
      S->q2 = S->Kp*S->Td/S->T;
      }

}

// funkce pro výpočet následujícího kroku u při nové hodnotě y
void pid_tick( pid_instance * S, float y )
{
      // posun hodnot e[(k-1)T] -> e[(k-2)T]
      S->e[2] = S->e[1];
      // posun hodnot e(kT) -> e[(k-1)T]
      S->e[1] = S->e[0];
      // posun hodnot new_e(kT) -> e(kT)
      S->e[0] = S->w - y;
```

```
	// posun hodnot u[(k-1)T] -> e[(k-2)T]
	S->u[1] = S->u[0];

	S->u[0] = S->u[1] + S->q0*S->e[0]  + S->q1*S->e[1] + S->q2*S->e[2];

	// omezeni výstupu
	S->u[0] = constrain(S->u[0],S->limit_min,S->limit_max);
}
```

Výše uvedeným zdrojovým kódem jsem realizoval číslicovou regulaci pomocí mikrokontroléru STM32 a testoval na úlohách řízení vybraných typů laboratorních systémů.

6.2 Ověření funkčnosti na RC členu

Pro ověření implementace PID regulátoru jsem zvolil nejjednodušší možnou soustavu se známým matematickým modelem a to RC člen.



Obr. 37 - Elektrické schéma RC členu.

V časové oblasti je model popsán vztahem:

$$u_2(t) = k_i \int_0^t u_1(t)dt. \tag{13}$$

V oblasti komplexní proměnné je model popsán vztahem:

$$U_2(s) = U_1(s)\frac{\frac{1}{sC}}{R+\frac{1}{sC}} = U_1(s)\frac{1}{sRC+1}, \tag{14}$$

$$\mathrm{G(s)} = \frac{U_2(s)}{U_1(s)} = \frac{1}{sRC+1} = \frac{1}{T_1 s+1}. \tag{15}$$

Jedná se o proporcionální soustavu se setrvačností 1. řádu. Reálná soustava měla parametry elektronických součástek rezistorem definovaný odporem o velikosti 10 kΩ a kondenzátorem s kapacitou 220 μF.

Identifikaci modelu jsem provedl při skokové změně vstupního napětí z $u = 0$ V na $u = 1{,}632$ V. Hodnota 1,632 V odpovídá polovině rozsahu použitých 12bitových DA a AD převodníků (2^{12}=4096) při maximálním napětí 3,26 V. Vzorkovací perioda mikroprocesoru byla nastavena na hodnotu 0,1 s.

Aproximace přechodové charakteristiky [SLOVÁK, RIEDEL, 2019]

Matematický model přenosu proporcionální soustavu se setrvačností 1. řádu bez dopravního zpoždění má tvar

$$\mathrm{G(s)} = \frac{k_1}{T_1 s + 1}, \tag{16}$$

kde hodnota parametru k_1, který reprezentuje zesílení soustavy, se vypočítá z hodnot vstupního u a výstupního y signálu soustavy v ustáleném stavu

$$y(\infty) = 1{,}632\ V, \qquad u(\infty) = 1{,}632\ V\ , \tag{17}$$

úroveň obou signálů je stejná a tak soustava signál nezesiluje a ani netlumí

$$k_1 = \frac{y(\infty)}{u(\infty)} = \frac{1{,}632}{1{,}632} = 1. \tag{18}$$

Zásadní hodnota, která formuje přechodovou charakteristiku, je úroveň výstupního signálu v 63 % hodnoty ustáleného stavu

$$0{,}63y(\infty) = 1{,}0282\ V, \tag{19}$$

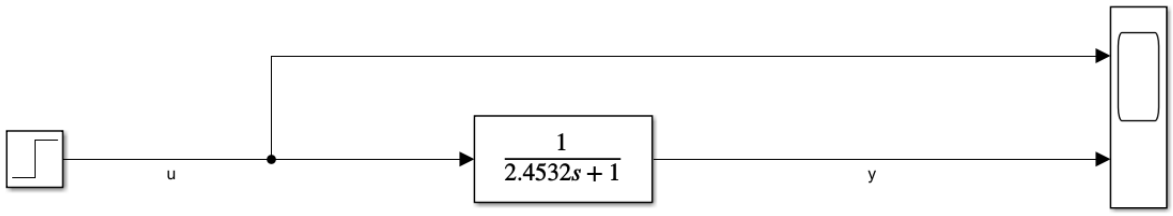
a T_1 je čas, ve kterém průběh dosáhne hodnoty 63% ustáleného stavu

$$T_1 \sim 0{,}63y(\infty) = 2{,}4532s. \tag{20}$$

Po dosazení do rovnice (16) bude výsledná aproximovaná přechodová charakteristika ve tvaru

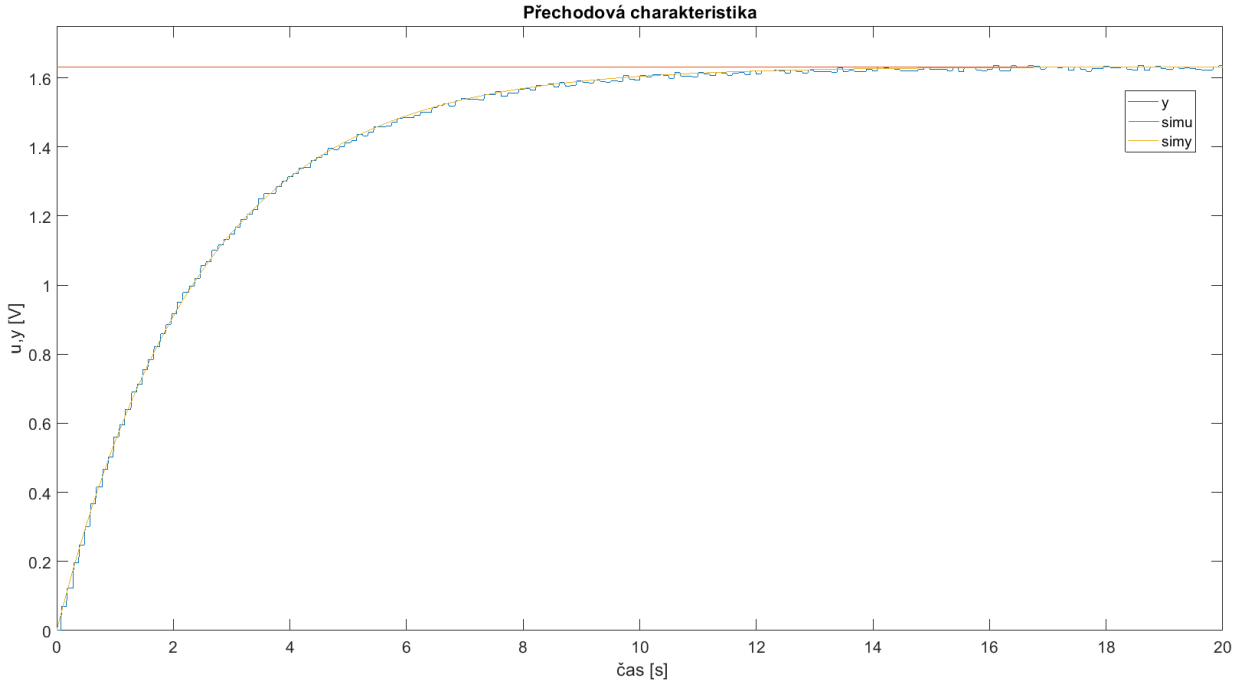
$$\mathrm{G(s)} = \frac{1}{2{,}4532s + 1}. \tag{21}$$

Pro ověření přesnosti identifikace jsem provedl simulaci v programu MATLAB & Simulink od společnosti MathWorks.



Obr. 38 - Simulační schéma pro ověření přesnosti identifikace přechodové charakteristiky.

Na Obr. 39 jsou zobrazeny průběhy výstupu systému ve tvaru přechodové charakteristiky označené signálem *y* pro signál z reálné soustavy a signálu *simy* pro simulovanou přechodovou charakteristiku (Obr. 38). Simulovaný vstupní signál ve tvaru skokové změny je v grafu (Obr. 39) s označením *simu*, který je shodný se skokovou změnou získanou z reálného RC členu označeného v grafu *u*. Na první pohled je patrné, že obě křivky *y* i *simy* jsou tvarem shodné.



Obr. 39 - Změřená a simulovaná přechodová charakteristika.

Nastavení parametrů PS regulátoru

Pro regulovanou soustavu

$$G_S(\mathrm{s}) = \frac{1}{RCs+1} = \frac{k_1}{T_1 s+1} = \frac{1}{2{,}4532s+1}, \tag{22}$$

jejíž parametry k_1 a T_1 byly identifikovány metodou aproximace proporcionální soustavy 1. řádu se setrvačností z přechodové charakteristiky získané z přechodové charakteristiky měřením s využitím mikrokontroléru STM32. Parametr $k_1 = 1$ a $T_1 = 2{,}4532$ s tvoří konkrétní RC člen jako regulovanou soustavu, pro niž byl navržen PS regulátor s hodnotou vzorkovací periody $T_1 = 0{,}1$ s. Pro proporcionální systém se setrvačností 1. řádu bez dopravního zpoždění je pro návrh parametrů PS regulátoru využita metoda požadovaného modelu.

Metoda požadovaného modelu

Jedná se o analyticko-experimentální metodu a je vhodná pro snadné a rychlé seřízení konvenčních regulátorů pro základní druhy regulovaných soustav i s dopravním zpožděním. Jedná se o metodu, která byla zpracována na Fakultě strojní VŠB – Technické univerzitě Ostrava [Vítečková, Víteček, 2011].

Optimální integrační konstantu T_I^* PS regulátoru vypočítáme dle vztahu

$$T_I^* = T_1 - \frac{T}{2} = 2{,}4532 - \frac{0{,}1}{2} = 2{,}4032\ [s] \tag{23}$$

a optimální hodnotu zesílení pro soustavu bez dopravního zpoždění určíme dle vztahu

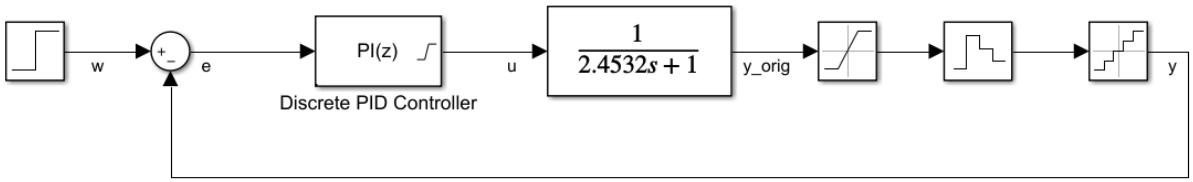
$$k_p^* = \frac{2T_I^*}{k_1(2T_w + T)}, \tag{24}$$

kde hodnota T_w se navrhuje s ohledem na omezení akční veličiny. Požadavkem kvality regulace je průběh regulované veličiny s maximálně 5% překmitem a nejkratší dobou regulace s požadavkem relativní tolerance regulace s hodnotou 0,05.

Prvotní návrh parametru $T_w = 2$, pak hodnota optimálního parametru k_p^* odvozená z přenosu řízení má závislost

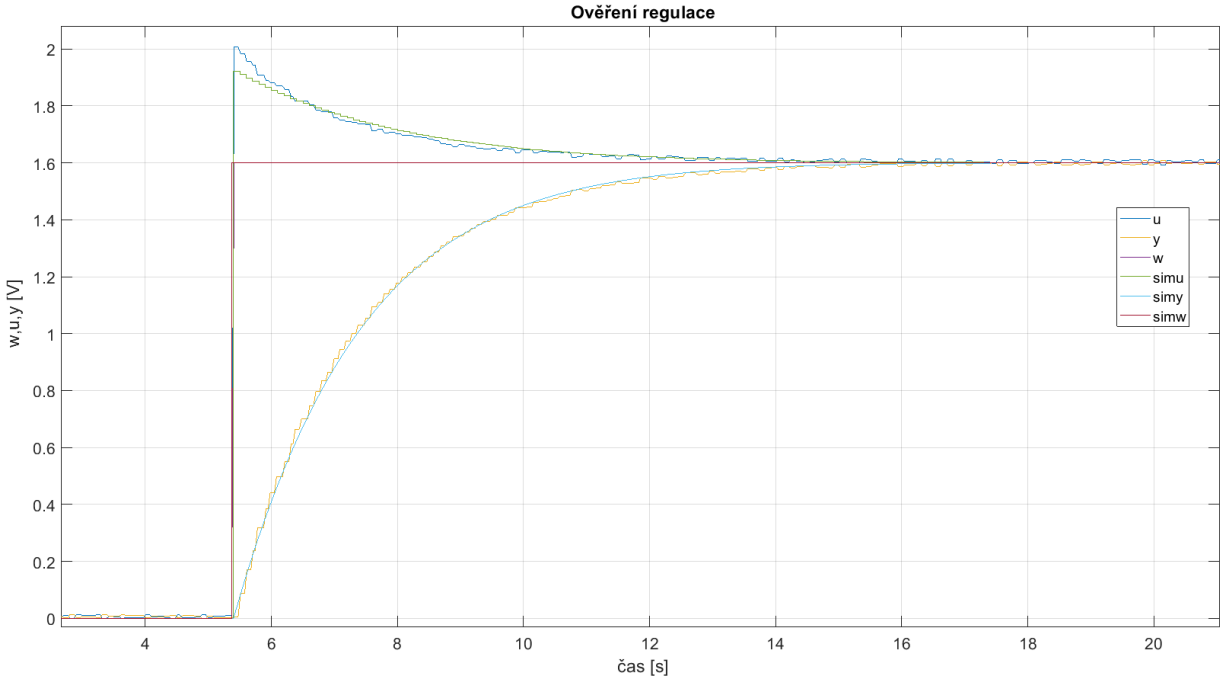
$$k_p^* = \frac{T_I^*}{k_1 T_w} = \frac{2{,}4032}{1 \cdot 2} = 1{,}2016\,. \tag{25}$$

Simulace regulačního procesu jsem provedl v simulačním programu MATLAB & Simulink a simulační schéma je na Obr. 40.



Obr. 40 - Simulační schéma pro program Matlab & Simulink.

V simulaci regulačního obvodu danou soustavou a výše navrženými parametry regulátoru $T_I^* = 2{,}4032$ a $k_p^* = 1{,}2016$ jsem ověřil, že doba regulace je $t_r = 5{,}759$ s a bez překmitu.

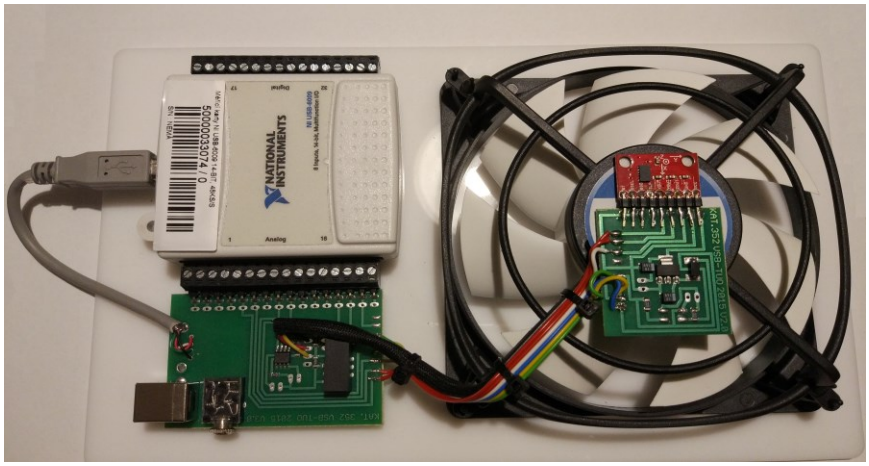


Obr. 41 – Simulovaný a změřený regulační proces v mikrokontroléru.

Ze simulovaných a naměřených průběhů je patrné, že jsou shodné a naprogramovaný regulátor funguje dobře. Je tedy možné přistoupit k realizaci regulace na laboratorním modelu ventilátoru.

7 Realizace připojení laboratorního modelu

Zvoleným laboratorním modelem je model ventilátoru (Obr. 42), u kterého je k dispozici měřicí karta od společnosti National Instruments NI USB-6009 nahrazena vývojovou deskou Nucleo-F767ZI.



Obr. 42 - Model ventilátoru.

7.1 Popis laboratorního modelu

Model (Obr. 42) slouží jako pomůcka pro hromadnou výuku studentů a demonstrují se na ní základní principy automatického řízení. Tomuto určení odpovídá i robustnost modelu a je pamatováno i na to, že student si mnohdy nevybaví potřebnou teoretickou přípravu k problematice a postupuje metodou pokus omyl, která by mohla vést k poškození modelu nebo úrazu studenta.

Řízený systém představuje ventilátor o průměru 11 cm, na který je připevněn akcelerometr a přídavná elektronika pro zajištění zvýšení napájecího napětí na 12 V. Otáčky ventilátoru se měří s využitím reflexního senzoru (LED dioda + fototranzistor) nebo pomocí Hallovy sondy integrované v těle ventilátoru. Napájecí napětí modelu je získáno z USB portu počítače.

7.2 Popis připojení laboratorního modelu k vývojové desce

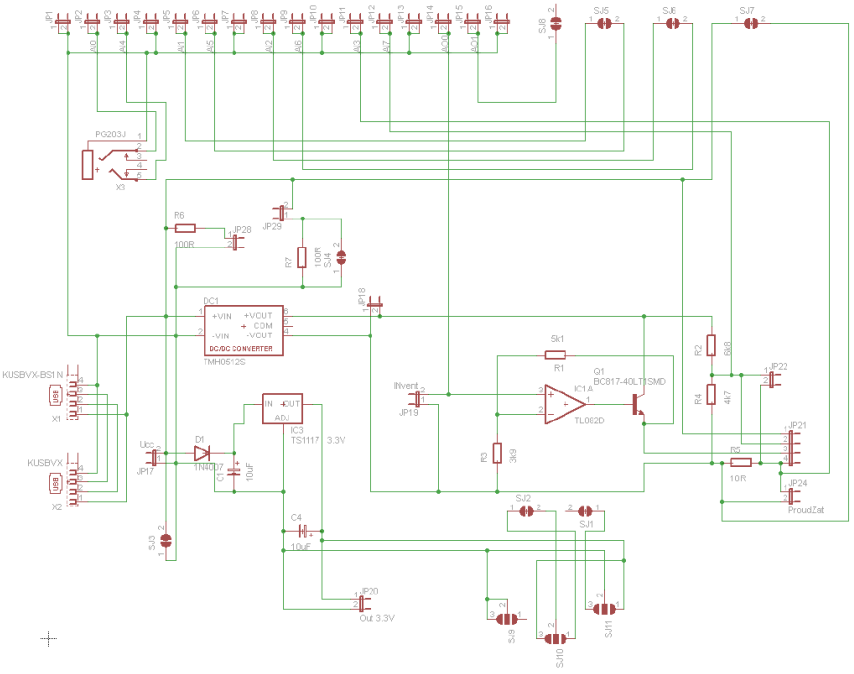
Vývojová deska je připojena k modelu pomocí jednoho analogového výstupu a jednoho analogového vstupu.

Analogový výstup vývojové desky s označením PA4, který je nastavený pro výstup DA převodníku – kanálu 1. Tento výstup je připojený na neinvertující vstup operačního zesilovače IC1A. Operační zesilovač je pomocí rezistorů R1 a R3 zapojený jako neinvertujícího zesilovače a

tranzistor Q1 na jeho výstupu slouží k proudovému posílení výstupu zesilovače. Tranzistor spíná výstupní napětí z DC-DC převodníku DC1, který zajišťuje zdroj 12 V, která násobí z napětí přivedeného z USB portu.

Analogový vstup vývojové desky s označením PA3, který je nastavený pro vstup AD převodníku ADC1/kanálu 3, slouží pro měření napětí z reflexního senzoru, který měří vzdálenost povrchu lopatky od detektoru senzoru. Tím je generovaný analogový signál, který slouží pro měření otáček.

Dalším způsobem, jak měřit otáčky je výstup ventilátoru, který má vestavěný Hallův senzor, generující obdélníkový signál, jehož frekvence se mění s rychlostí otáčení ventilátoru.



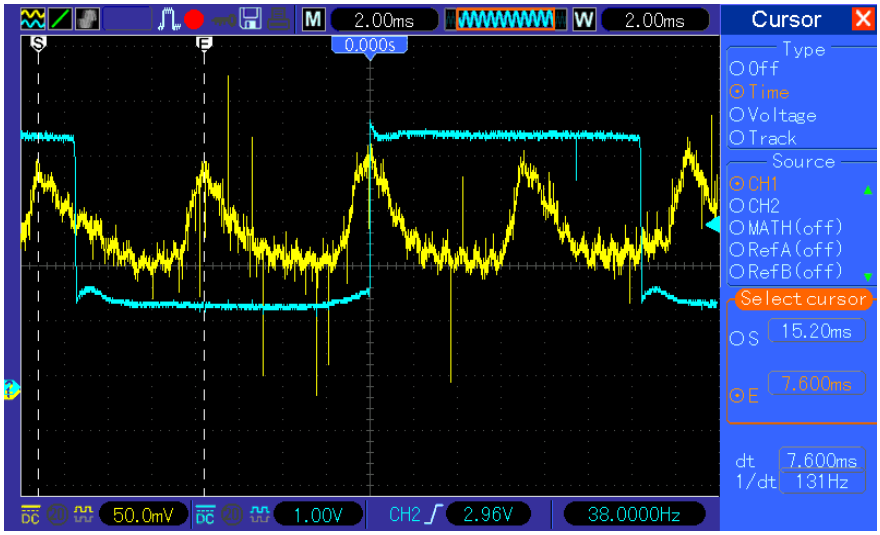
Obr. 43 Schéma obslužné elektroniky modelu [KULHÁNEK, ŠKUTA, 2015].

7.3 Popis ovládání laboratorního modelu

Otáčky ventilátoru jsou nastavovány pomocí elektrického napětí v rozsahu 1 až 5 V. Zdrojem tohoto napětí je DA převodník, který je součástí mikroprocesoru na vývojové desce Nucleo. Avšak DA převodník pracuje s výstupním elektrickým napětím 0 až 3,3 V a proto je použit zesilovač s operačním zesilovačem v neinvertujícím zapojení, který zesílí napětí 0 až 3,3 V na 0 až 5 V.

Model se pro napětí 0 – 1 V chová nedeterministicky, proto je hodnota výstupního napětí DA převodníku ponížena o 1000 jednotek (0,795 V), ať je tato oblast vyloučena z ovládání a ventilátor je ovladatelný od nulového napětí.

Otáčky ventilátoru lze měřit z šířky pulzů, které přímo generuje ventilátor nebo pomocí měření signálu z reflexního senzoru. Reflexního senzor poskytuje přesnější informaci o aktuálních otáčkách. Ventilátor má 7 lopatek a tak je změna otáček znatelnější.



Obr. 44 - Měření signálu otáček osciloskopem.

Na Obr. 44 jsou vidět dva průběhy. Modrý je signál od Hallovy sondy z ventilátoru a žlutý je signál od reflexního senzoru, který je uchycen na těle ventilátoru.

Signál z reflexního senzoru se pohybuje mezi 50 a 170 mV a jeho frekvence se pohybuje od 80 do 140 Hz. Tento signál má též na sobě namodulovaný šum a tak je práce s ním značně složitá.

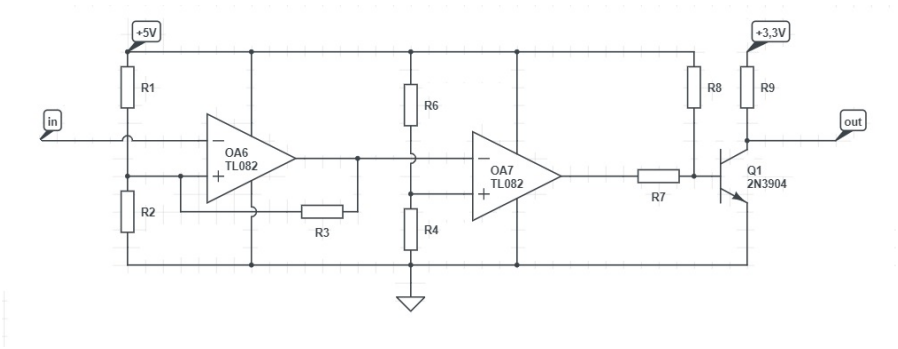
Výpočet otáček z periody signálu reflexního senzoru

$$otáčky \ = 60 \cdot 7 \cdot \frac{1}{T} \ [rad^{-1}, s]. \tag{26}$$

Otáčky za minutu se vypočítají jako převrácená hodnota periody, tím získáme frekvenci, kterou následně vynásobíme 60, ať získáme ze sekund minuty a ještě vynásobíme 7, ať zohledníme, že počet lopatek je 7.

Ventilátor má v sobě integrovanou elektroniku, která pomocí Hallovy sondy, při každé čtvrtině otáčky, invertuje výstup. Tato elektronika potřebuje napájení, proto se na statické charakteristice projevuje necitlivost v nízkých napájecích napětích. Úkolem této elektroniky je též omezení maximálních otáček, proto se při vyšších napětích projevuje saturace.

Signál pro měření otáček z ventilátoru je obdélníkový s amplitudou 5 V a jeho frekvence se pohybuje od 24 po 40 Hz. Avšak tento signál neodpovídá TTL logice, jelikož jeho v úrovni logické 0 je okolo 1,7 V a ani rozsahu 0 až 3,3 V, se kterými pracují vstupy mikrokontroléru. Proto je potřeba signál upravit a pomocí obvodu na Obr. 45.



Obr. 45 - Obvod pro zpracování signálu ventilátoru.

Obvod má tři části. První, která je tvořena operačním zesilovačem OA1, je Schmittův klopný obvod, který funguje jako invertující zesilovač s hysterezí. Druhá část, která je tvořena OA2, je invertující zesilovač, který pouze invertuje signál, ať je stejný jako na vstupu. Třetí část, která je tvořena tranzistorem Q1, funguje jako spínač napětí 3,3 V a tím upraví signál na úroveň TTL.

V programu je měření otáček zajištěno pomocí čítače s externím přerušením, jeho nastavení je popsáno v kapitole 5.8.

Pro vlastní měření šířky pulzu jsem naprogramoval tento kód:

```
// preruseni od countru pro mereni sirky pulzu z externiho zdroje
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
	if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1)   // pokud je zdrojem přerušení kanál časovače 1
	{
	if (Is_First_Captured==0)  // nastalo první volání – naběžná hrana ?
	{
	IC_Val1 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1); // vyčtení první hodnoty čítače
	Is_First_Captured = 1;  // není první vyčetení
	// změna polarity externího signálu na sestupnou hranu
```

```
      __HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_1,
TIM_INPUTCHANNELPOLARITY_FALLING);
      }
      else if (Is_First_Captured==1)   // pokud je sestupná hrana
      {
      IC_Val2 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);   // vyčteme
druhou hodnotu čítače
      __HAL_TIM_SET_COUNTER(htim, 0);  // zresetujeme čítač

      if (IC_Val2 > IC_Val1)
      {
      Difference = IC_Val2-IC_Val1; // vypočítáme dobu trvání pulzu
      }

      Is_First_Captured = 0; // znulujeme příznak nového pulzu

      // nastavíme polaritu externího pulzu na náběžnou hranu
      __HAL_TIM_SET_CAPTUREPOLARITY(htim,                    TIM_CHANNEL_1,
TIM_INPUTCHANNELPOLARITY_RISING);
      }

      RPM = (1500*1000)/Difference; // výpočet otáček
      RPM_a = average( RPM, 40 ); // výpočet klouzavého průměru otáček
      RPM_a2 = SampleAvgNDeleteX(40 , 12); // vynechání náhodných chyb z
průměrování
      }
}
```

Protože hodnota vypočítaných otáček hodně kolísá, napsal jsem funkci `average()`, která vypočítá klouzavý průměr, z hodnot udaných v parametru `pocet` a nová hodnota je předána pomocí parametru `new_in`, který zařadí tuto hodnotu do pole a ostatní posune. To zajistí vetší stabilitu hodnoty měření.

```
uint16_t hodnoty[40];
uint16_t average( uint16_t new_in, uint8_t pocet ){
```

```
	uint8_t i;
	uint32_t tmp = 0;

	for(i=pocet-1;i>0;i--) {
	hodnoty[i]=hodnoty[i-1];
	tmp += hodnoty[i];
	}
	hodnoty[0]=new_in;
	tmp += hodnoty[0];

	return (uint16_t) (tmp/pocet);
}
```

Pro zlepšení výsledku klouzavého průměru jsem z průměrování vynechal prvních a posledních 6 hodnot ze seřazeného souboru hodnot pomocí funkce `SampleAvgNDeleteX()`. Parametry funkce jsou počet hodnot souboru a počet vynechaných hodnot. Funkce má následující kód:

```
uint16_t SampleAvgNDeleteX(uint8_t N , uint8_t X)
{
	// N - počet prvku
	// X - počet prvku, které se odstarní ze seřazeného souboru z obou konců.
Po pulce z každé strany
	uint32_t avg_sample =0x00;
	uint8_t index=0x00;

	// překopírování souboru hodnot do souboru hodnot k seřazení
	for (index=0;index<N;index++) hodnoty_tmp[index] = hodnoty[index];
	/* seřazení souboru hodnot */
	Sort_tab(hodnoty_tmp,N);
	/* nevyloučené hodnoty jsou zprůměrovány */
	for (index=X/2; index<N-X/2; index++)
	{
	avg_sample += hodnoty_tmp[index];
	}
```

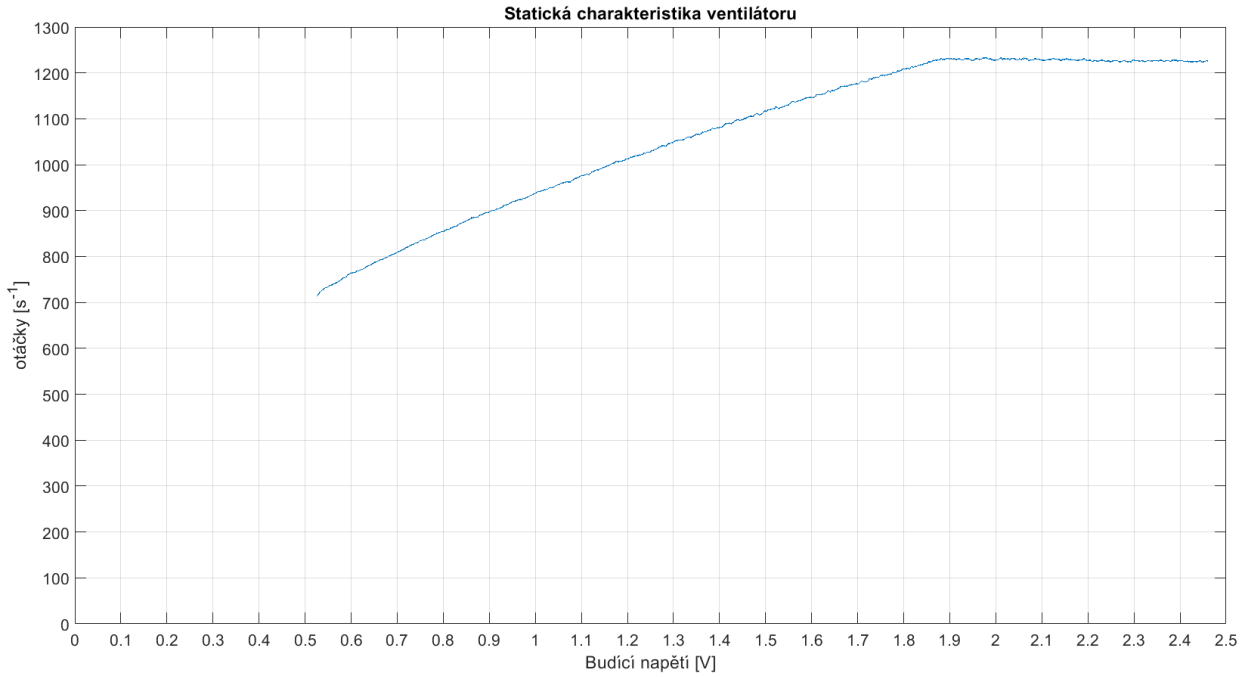
```
      /* výpočet průměru s vynecháním vzorků, které by mohly být náhodnými
chybami*/
      avg_sample /= N-X;
      /* vrácení  */
      return avg_sample;
}
```

Pro vyhodnocení souboru hodnot a odstranění náhodných chyb je třeba soubor hodnot seřadit, to je provedeno funkcí `Sort_tab()`, které jsou jako parametry předány soubor hodnot a délka souboru. Funkce má kód:

```
void Sort_tab(uint16_t tab[], uint8_t lenght)
{
  uint8_t l=0x00, exchange =0x01;
  uint16_t tmp=0x00;
  /* Sort tab */
  while(exchange==1)
  {
    exchange=0;
      for(l=0; l<lenght-1; l++)
      {
        if( tab[l] > tab[l+1] )
        {
          tmp = tab[l];
          tab[l] = tab[l+1];
          tab[l+1] = tmp;
          exchange=1;
        }
      }
  }
}
```

Statickou charakteristiku jsem měřil pomocí smyčky v programu, kdy jsem zvýšil hodnotu výstupu DA převodníku o jeden dílek a počkal 500 ms, až dojde k ustálení hodnoty.

Ze statické charakteristiky ventilátoru na Obr. 46, je patrný její nelineární charakter. Do napětí 0,5 V se projevuje necitlivost a od napětí 1,9 V se projevuje saturace.



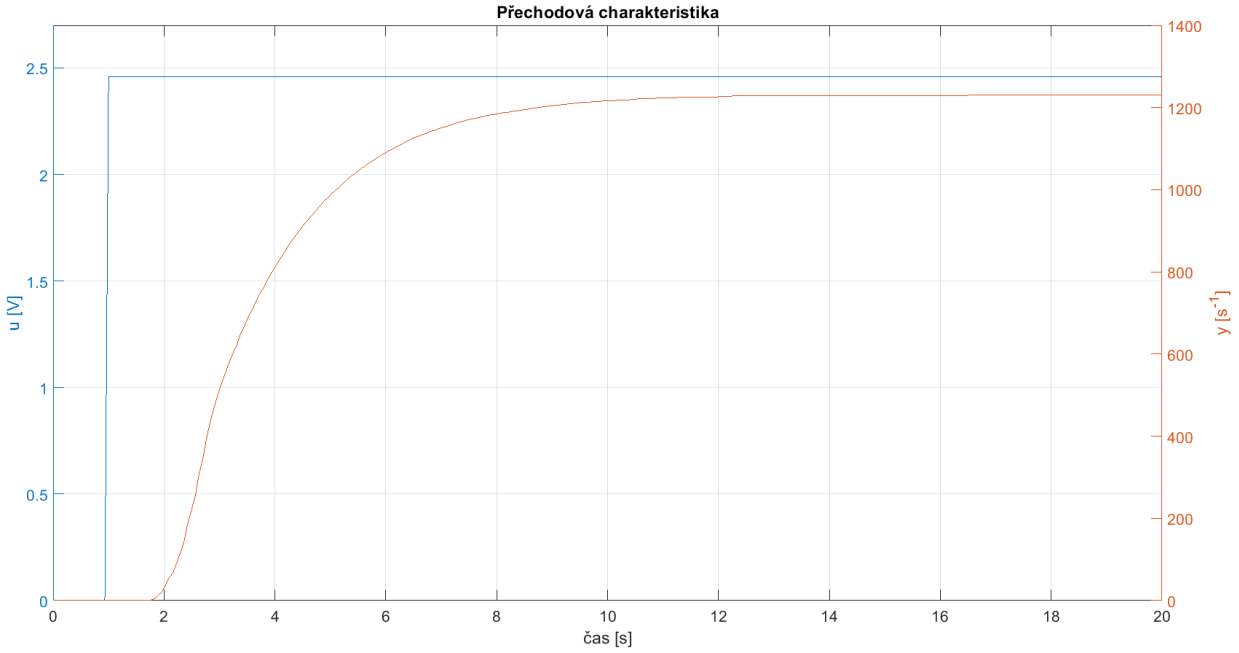
Obr. 46 - Statická charakteristika ventilátoru.

7.4 Regulace otáček ventilátoru

Přípravou pro řízení je znalost matematického modelu systému, který se získá identifikací přechodové charakteristiky. Přechodová charakteristika je odezva systému na skokovou změnu vstupního signálu.

Pro měření přechodové charakteristiky jsem zvolil vstupní napětí $u = 2{,}46$ V, kterému odpovídají otáčky ventilátoru $y \approx 1231$ s^{-1}. Lineární část statické charakteristiky má rozsah vstupního napětí u od 0,53 V do 1,9 V, kterému odpovídají otáčky ventilátoru y v rozsahu 720 s^{-1} až 1230 s^{-1}.

Na Obr. 47 je změřená přechodová charakteristika. Tvar odpovídá proporcionálnímu sytému se setrvačností druhého řádu s dopravním zpožděním.



Obr. 47 - Změřená přechodová charakteristika.

Identifikaci systému jsem provedl pomocí aproximace změřené přechodové charakteristiky tříbodovou metodou přenosem ve tvaru [SLOVÁK, RIEDEL, 2019]:

$$G_s(\mathrm{s}) = \frac{k_1}{(\mathrm{T_1 s}+1)(\mathrm{T_2 s}+1)} e^{-T_d s}. \tag{27}$$

Z grafu přechodové charakteristiky (Obr. 47) odečteme hodnoty $y_{0,7} = 0{,}7y(\infty)$, $y_{0,26} = 0{,}26y(\infty)$, $y_{0,09} = 0{,}09y(\infty)$ a jim odpovídající časy $t_{0,7}$, $t_{0,26}$, $t_{0,09}$.

$$y_\infty = 1231, \tag{28}$$

$$y_{0,7} = 0{,}7y_\infty = 0{,}7 \cdot 1231 = 862\ s^{-1} \Rightarrow t_{0.7} = 3{,}23\ s, \tag{29}$$

$$y_{0,26} = 0{,}26y_\infty = 0{,}26 \cdot 1231 = 320\ s^{-1} \Rightarrow t_{0.26} = 1{,}66\ s, \tag{30}$$

$$y_{0,09} = 0{,}09y_\infty = 0{,}09 \cdot 1231 = 111\ s^{-1} \Rightarrow t_{0.09} = 1{,}28\ s. \tag{31}$$

Dopravní zpoždění T_d určíme podle vztahu

$$T_d = 2\,t_{0,09} - t_{0,26} = 2 \cdot 1{,}28 - 1{,}66 = 0{,}9\ s. \tag{32}$$

Vypočítáme koeficienty B a C podle vzorců

$$\begin{aligned} B &= 0{,}83 \cdot t_{0,7} - 0{,}24 \cdot t_{0,26} + 0{,}48 \cdot t_{0,09} - T_d \\ &= 0{,}83 \cdot 3{,}23 - 0{,}24 \cdot 1{,}66 + 0{,}48 \cdot 1{,}28 - 0{,}9 = 1{,}9959, \end{aligned} \tag{33}$$

$$C = 4 \cdot \left(t_{0,26} - t_{0,09}\right)^2 = 4 \cdot (1{,}66 - 1{,}28)^2 = 0{,}5663. \tag{34}$$

Konstanty k_1, T_1 a T_2 určíme podle vztahů

$$T_1 = \frac{B + \sqrt{B^2 - 4 \cdot C}}{2} = \frac{1{,}9959 + \sqrt{1{,}9959^2 - 0{,}5663}}{2} = 1{,}6534\ s, \tag{35}$$

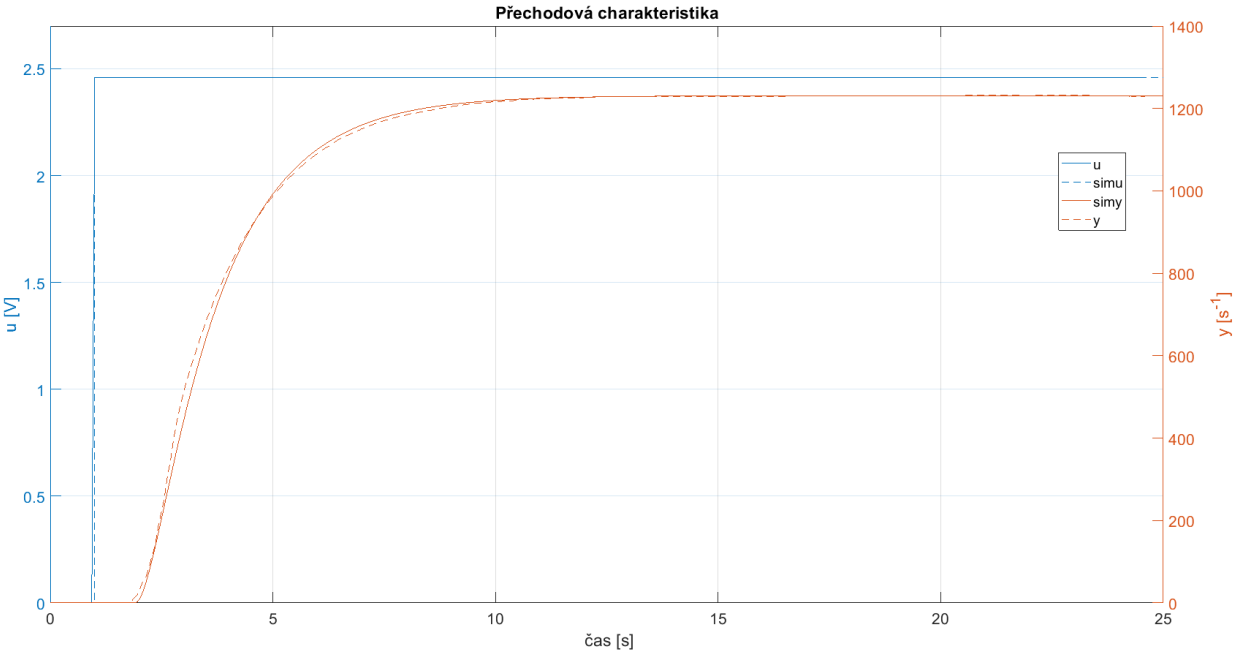
$$T_2 = \frac{B - \sqrt{B^2 - 4 \cdot C}}{2} = \frac{1{,}9959 - \sqrt{1{,}9959^2 - 0{,}5663}}{2} = 0{,}3425\ s, \tag{36}$$

$$k_1 = \frac{y(\infty)}{u(\infty)} = \frac{1231}{2{,}46} = 500{,}41. \tag{37}$$

Dosazením do vztahu (14) pro náhradní přenos získáme výsledný přenos proporcionální regulované soustavy se setrvačností 2. řádu s dopravním zpožděním:

$$G_s(\mathrm{s}) = \frac{k_1}{(\mathrm{T_1 s} + 1)(\mathrm{T_2 s} + 1)} e^{-T_d s} = \frac{500{,}41}{(1{,}6534\mathrm{s} + 1)(0{,}3425\mathrm{s} + 1)} e^{-0{,}9s} \tag{38}$$

Pomocí programu MATLAB & Simulink jsem provedl simulaci odezvy aproximovaného přenosu (38) na skokovou změnu vstupního signálu z 0 na 2,46 V. Graf průběhu naměřené a simulované přechodové charakteristiky je na Obr. 48.



Obr. 48 - Přechodová charakteristika změřená a simulovaná.

Pro identifikovaný systém nyní navrhnu regulátor pomocí metody požadovaného modelu.

Metoda požadovaného modelu

Metoda je popsána dříve a to v kapitole 6.2, při ověřování funkčnosti kódu regulátoru. Nyní využiji tuto metodu pro seřízení parametrů regulátoru pro soustavu ventilátoru. Identifikací byl získán přenos soustavy ve tvaru

$$G_s(\mathrm{s}) = \frac{k_1}{(\mathrm{T_1 s}+1)(\mathrm{T_2 s}+1)} e^{-T_d s} = \frac{500{,}41}{(1{,}6534\mathrm{s}+1)(0{,}3425\mathrm{s}+1)} e^{-0{,}9s} \tag{39}$$

Pro soustavu s tímto přenosem metoda navrhuje použití PSD regulátoru s přenosem

$$G_R(z) = k_p \left(1 + \frac{T}{T_I}\frac{z}{z-1} + \frac{T_D}{T}\frac{z-1}{z}\right). \tag{40}$$

Optimální integrační časovou konstantu T_I^* PS regulátoru vypočítáme dle vztahu

$$T_I^* = T_1 + T_2 - T = 1{,}6534 + 0{,}3425 - 0{,}1 = 1{,}8959\ [s], \tag{41}$$

optimální derivační časovou konstantu T_D^* PS regulátoru vypočítáme dle vztahu

$$T_D^* = \frac{T_1 \cdot T_2}{T_1 + T_2} - \frac{T}{4} = \frac{1{,}6534 \cdot 0{,}3425}{1{,}6534 + 0{,}3425} - \frac{0{,}1}{4} = 0{,}2587\ [s] \tag{42}$$

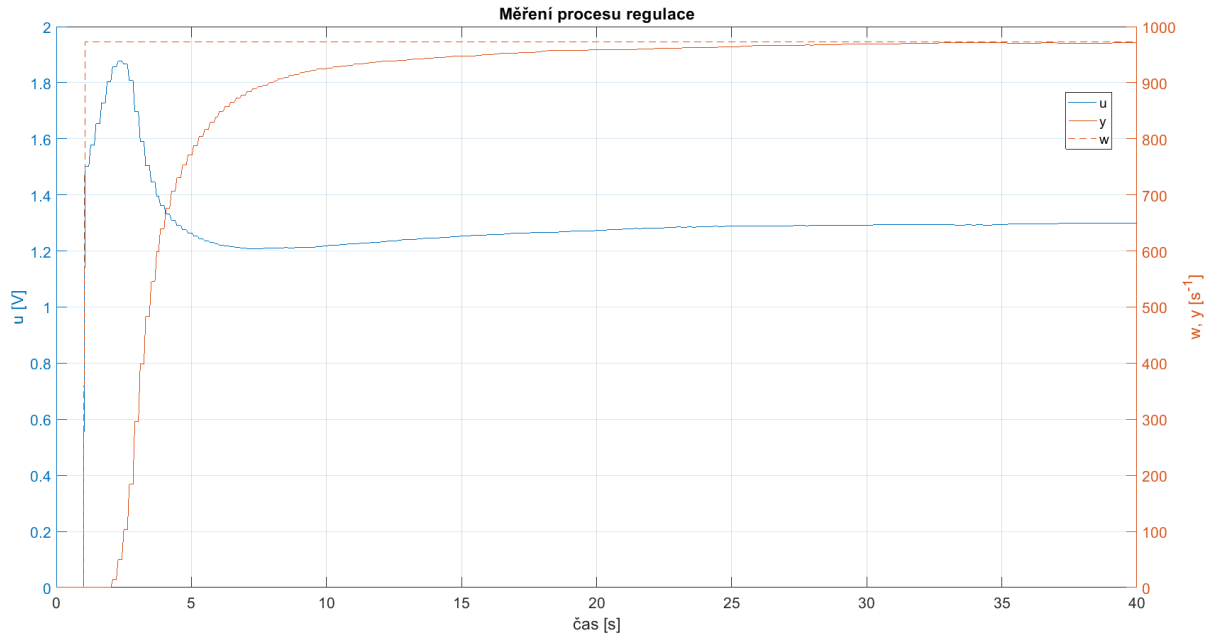
a optimální hodnotu zesílení pro soustavu bez dopravního zpoždění určíme dle vztahu (43), kdy hodnoty koeficientů α a β pro relativní překmit $\kappa = 0$ % jsou $\alpha = 1{,}282$ a $\beta = 2{,}718$. Hodnota T_w je pro prvotní návrh zvolena na hodnotu $T_w = 2$, pak zesílení je dáno

$$k_p^* = \frac{T_I^*}{k_1(\alpha T_w + \beta T_d)} = \frac{1{,}8959}{500{,}4065 \cdot (1{,}282 \cdot 2 + 2{,}718 \cdot 0{,}9)} = 0.001467. \tag{43}$$

Tímto jsou všechny koeficienty vypočítané a výsledný přenos má, pro vzorkovací periodu T = 0,1 s tvar

$$G_R(z) = k_p \left(1 + \frac{T}{T_I}\frac{z}{z-1} + \frac{T_D}{T}\frac{z-1}{z}\right) = 0.001467 \cdot \left(1 + \frac{0{,}1}{1{,}8959}\frac{z}{z-1} + \frac{0{,}2587}{0{,}1}\frac{z}{z-1}\right). \tag{44}$$

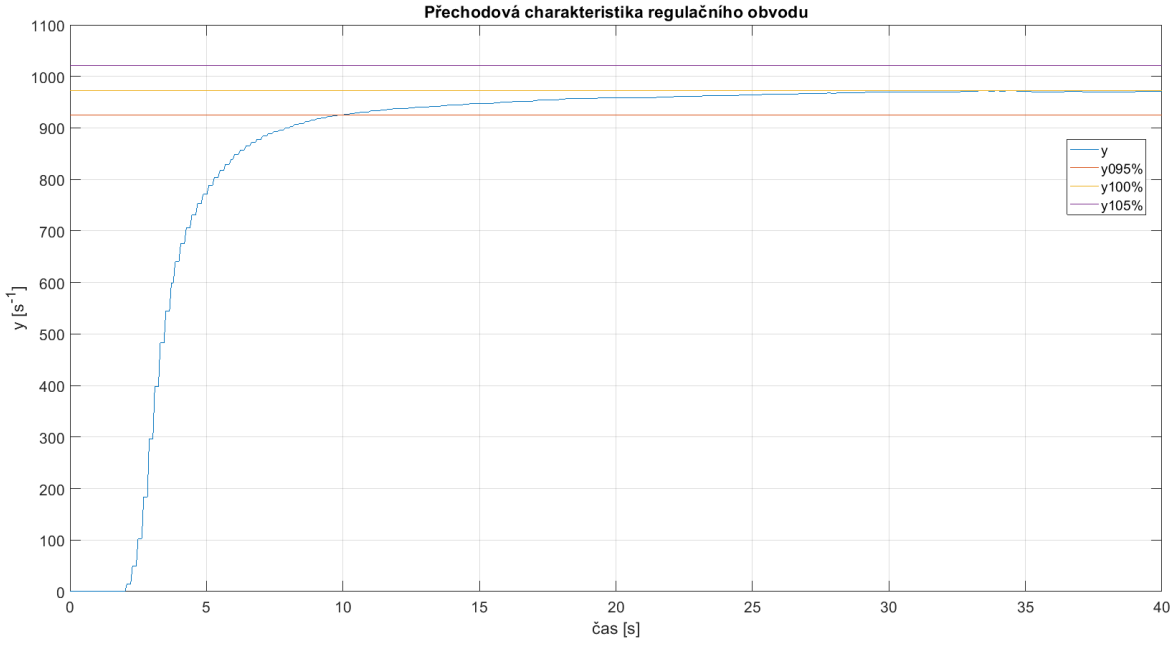
Proces regulace jsem ověřil pro skokovou změnu žádané veličiny z 0 na 973 s^{-1}. Průběh žádané veličiny $w(t)$, akčního zásahu $u(t)$ a regulované veličiny $y(t)$ je zobrazen v grafu na Obr. 49.



Obr. 49 - Změřený proces regulace.

Graf na Obr. 50 zobrazuje přechodovou charakteristiku regulačního obvodu s vyznačenými ukazateli kvality, kterými jsou pásmo $y(\infty) \pm \Delta$, kdy $\Delta = \delta y(\infty)$ a $\delta = 0{,}01 \div 0{,}05$. [Vítečková, Víteček, 2008]

Doba regulace je t_r = 8,8 s bez relativního překmitu κ.



Obr. 50 - Přechodová charakteristika regulačního obvodu s vyznačenými ukazateli kvality.

8 Závěr

V rámci své práce jsem se seznámil s rodinou mikrokontrolerů STM32 výrobce STMicroelectronics. Tyto mikroprocesory jsou postavené na RISC technologii a jejich jádro designované firmou ARM. Celá tato rodina má 12 podskupin, které jsou členěný podle plánovaného užití, výkonosti a vybavenosti. Podrobněji jsem se zaměřil na podskupinu STM32F7, která obsahuje výkonné mikroprocesory postavené na technologii ARM Cortex-7.

Obecné seznámení s řadou mikroprocesorů STM32F7 bylo přípravou pro další krok a tím bylo seznámení se s vývojovou deskou Nucleo-F767ZI, která slouží jako přípravný balíček pro prozkoumání možností práce a nasazení mikrokontrolerů STM32F7 v různých aplikacích. Vývojová deska je osazena mikroprocesorem STM32F767ZI.

Pro realizaci řízení procesů je třeba vytvořit řídicí program, který bude zajišťovat potřebné kroky řízení. K tomuto účelu bylo třeba se seznámit s dostupnými variantami vývojových prostředí, která jsou pro programování aplikací vhodné a přináší patřičný komfort pro vývoj a ladění vyvíjené aplikace. Z tohoto důvodu jsem provedl rozbor dostupným vývojových prostředí: KEIL µVision, Atollic TrueSTUDIO, STM32Cube IDE a System Workbench for STM32, z nichž jsem zvolil KEIL µVision. Následně při konstrukci testovacích rozhraní potřebných pro řízení procesu i programem STM32CubeMX, která umožnuje sestavení kostry programu a přednastavení používaných periferií. Pro vizualizaci je vhodný program STM Studio, který umožnuje sledovat obsah proměnných a též graficky sledovat průběh jejich změn.

Program pro mikrokontrolér jsem programoval pro chod v jedné smyčce, kde využívám systém přerušení pro zpracování dat z periferií, nebo bude běžet v operačním systému, který umožní chod sdílení procesorového času formou časového multiplexu. Z tohoto důvodu jsem se v další části práce zaměřil na zjištění možností využití RTOS, který by se dalo využít pro realizaci řízení procesu. Seznámil jsem se s možnostmi využití RTOS Keil CMSIS-RTOS RTX, FreeRTOS a ChibiOS/RT. Jako nejzajímavější se jevil CMSIS-RTOS RTX, který je integrován do programu STM32CubeMX a i vývojového prostředí KEIL µVision.

Pro realizaci řízení procesu jsem se seznámil a sepsal testovací úlohy, které obsahují programování obsluhy AD převodníku a to v režimu konverzi analogového signálu na vyžádání, konverzi analogového signálu na základě podnětu od časovače a získání dat z více analogových kanálů na základě časovače a odeslání výsledku pomocí mechanizmu DMA do paměti.

Programování obsluhy DA převodníku, ovládání digitálních vstupů a výstupů. Dále jsem se seznámil s ovládáním časovače a spojením časovače s analogovým převodníkem pro získání dat s pevnou vzorkovací periodou.

Popsal jsem a napsal funkci v programovacím jazyku C pro implementaci číslicového regulátoru PID pomocí přírůstkového zápisu a napsal testovací pro ověření funkčnosti, kterou jsem vyzkoušel na modelu RC článku a experimentálním nastavením parametrů regulátoru.

Seznámil jsem se s laboratorním modelem ventilátoru až na úroveň pochopení schématu zapojení pomocných elektronických obvodů. Navrhnul způsob, jak jej spojit s vývojovou deskou Nucleo-F767ZI a připravil se na její řízení.

Pro propojení vývojové desky s laboratorním modulem jsem musel navrhnout a zkonstruovat obvody pro úpravu signálu. Jedná se o obvod zpracovávající signál, který je generovaný Hallovým senzorem, umístěném v těle ventilátoru. Tento signál je obdélníkový, avšak jeho nižší úroveň se průběhu změny otáček mění a dostane se mimo rozhodovací napěťovou úroveň binární logiky mikroprocesoru a ten nedokáže určit, zda je signál stále v úrovni logická 0. Tento obvod se skládá ze Schmittova klopného obvodu, který zajistí, že při změně úrovně nedojde k nechtěným zákmitům upraveného signálu. Schmittův klopný obvod jsem realizoval pomocí operačního zesilovače, avšak provádí inverzi signálu, a proto jsem tento signál následně upravil invertujícím zesilovačem se zesílením A_u=1. Signál vyhovuje TTL logice, která pracuje s napětím 5V pro úroveň logická 1, avšak mikrokontrolér pracuje v CMOS logice s napěťovou úrovní logické 1 o velikosti 3,3 V. Z toho důvodu jsem zařadil na výstup invertujícího zesilovače ještě tranzistor v zapojení spínače, který převede signál do úrovní logické nuly o napětí 0V a logické jedničky o napětí 3,3 V. Obdobnou úpravu jsem realizoval i pro signál generovaný DA převodníkem mikroprocesoru, protože laboratorní model je navržen pro práci s napětím 0-5V a mikrokontorolér má maximální napětí DA převodníku ve výši 3,3 V. Tento signál jsem upravil neinvertujícím zesilovačem realizovaným pomocí operačního zesilovače.

Otáčky ventilátoru měřím čítačem, který měří délku pulzů generovaných ventilátorem. Takto získaná hodnota je značně proměnlivá a tak na ni aplikuji klouzavý průměr s odstraněním 40% hodnot, které považuji za náhodné chyby s ohledem na normální rozložení měřených hodnot.

Oproti původně plánované realizaci pomocí RTOS se ukázalo, že výpočetní výkon mikrokontroléru je dostatečně velký a není potřeba využívat časový multiplex a přiřazení priority vykonávaných procedur. Naprosto postačuje vykonávání celého řídicího kódu pomocí přerušení.

Zásadními jsou dvě přerušení - čítače a časovače. V přerušení čítače, ve kterém se měří šířka pulzů od ventilátoru, a na základě nich jsou určeny výpočtem otáčky ventilátoru. Dalším přerušením je přerušení časovače, které generuje vzorkovací periodu PID regulátoru a hodnoty akčního zásahu pomocí DA převodníku.

Pomocí takto připraveného kódu jsem byl schopen změřit přechodovou charakteristiku ventilátoru a tím z jejího tvaru zjistit, že se jedná o proporcionální systém se setrvačností druhého řádu s dopravním zpožděním. Změřenou přechodovou charakteristiku jsem následně identifikoval pomocí tříbodové metody a tím získal matematický model ventilátoru v podobě přenosu. Na základě znalosti matematického modelu sytému jsem provedl syntézu regulačního obvodu pomocí metody požadovaného modelu. Metoda doporučila využití PID regulátoru pro řízení soustavy a na jejím základě jsem vypočítal parametry regulátoru pro jeho vhodné seřízení. Následně jsem provedl ověření těchto parametrů pomocí skokové změny žádané hodnoty a změřil odezvu regulačního obvodu a průběhy žádané veličiny a akčního zásahu. Zde můžu konstatovat, že celý regulační obvod funguje správně a mikrokontrolér s implementací PID regulátoru, kterou jsem programově navrhnul, je použitelný pro řízení modelu.

Obecně je možné prohlásit, že mikrokontoléry rodiny STM32F7 jsou vhodné pro řízení procesů. Na možná úskalí je možné narazit při zpracování vstupních signálů, které často obsahují rušení a mají nízkou úroveň napětí. Toto je i případ signálu od reflexního senzoru, kdy byl schopen provést jeho digitalizaci pomocí AD převodníku, avšak i přes to, že převodním je 12bitový jsem nedokázal silně zašumělý signál zpracovat. Aplikací klouzavého průměru s odstraněním náhodných chyb, došlo k částečnému vyhlazení signálu, ale zároveň k jeho utlumení, že jsem nebyl schopný softwarově rozhodnout, že došlo k průchodu lopatky kolem reflexního senzoru.

Jako další směr řešení vidím implementaci rozhraní Ethernet, které umožní rozšíření možností pro propojení vývojové desky s počítačem a realizaci prediktivního řízení na základě znalosti matematického modelu systému a případně realizaci řízení pomocí neuronové sítě, případně umělé inteligence. Dalším směrem řešení by mohla být implementace řízení modelu ventilátoru pomocí dat z reflexního senzoru, čím dojde k zpřesnění měření otáček modelu. STM32F7 obsahuje akceleraci FIR a IIR filtrů, které by se daly využít při zpracování signálu od senzoru.

9 Použitá literatura

ARM. RTX v5 Implementation [online]. [cit. 2018-04-07]. Dostupné z: http://arm-software.github.io/CMSIS_5/RTOS2/html/rtx5_impl.html

ARM KEIL. CMSIS-RTOS Keil RTX [online]. [cit. 2018-04-07]. Dostupné z:http://www2.keil.com/mdk5/cmsis/rtx

ARM KEIL. µVision® IDE [online]. [cit. 2018-04-07]. Dostupné z: http://www2.keil.com/mdk5/uvision/

ARM KEIL. RTX Real-Time Operating System [online]. [cit. 2018-04-07]. Dostupné z: http://www.keil.com/arm/rl-arm/kernel.asp

Circuit Basics. Basics of UART Communication [olnine]. [cit. 2019-10-21]. Dostupné z: http://www.circuitbasics.com/basics-uart-communication/

Di Sirio, Giovanni. ChibiOS/RT [online]. [cit. 2018-04-07]. Dostupné z: http://www.chibios.org/dokuwiki/doku.php?id=chibios:product:rt:start

DUDKA, Michal Úvod k STM32 [online]. [cit. 2018-04-07]. Dostupné z: http://www.pd-nb.cz/clanky/stm32_uvod/stm32_uvod.html

HEROUT, Pavel. Učebnice jazyka C. České Budějovice: Nakladatelství KOPP, 270s. ISBN 80-85828-21-9.

CHRASCINA, Václav. Programová podpora periferií mikroprocesorů řady STM32F4. Ostrava, 2015. Bakalářská práce. Katedra automatizační techniky a řízení, Fakulta strojní, VŠB-Technická univerzita Ostrava. Vedoucí práce Ing. David Fojtík, Ph.D.

KULHÁNEK, Jiří a Jaromír ŠKUTA. Design of Education Model with NI LabView. In: Proceedings of the 2015 16th International Carpathian Control Conference, ICCC 2015. New York: Institute of Electrical and Electronics Engineers, 2015. s. 267 - 270. ISBN 978-1-4799-7370-5.

LANFREY, Jean-Baptiste. Simulink to STM32 [online]. [cit. 2018-04-07]. Dostupné z: https://www.mathworks.com/content/dam/mathworks/mathworks-dot-com/company/events/conferences/matlab-tour-australia/2014/proceedings/simulink-to-stm32.pdf

SLOVÁK, Tomáš a RIEDL, Zdeněk. Identifikace a syntéza řízení technologických procesů [online]. [cit. 2019-11-16]. Dostupné z: http://books.fs.vsb.cz/Identifikace/index.htm

SROVNAL, Vilém. Operační systémy pro řízení v reálném čase. Ostrava: Vysoká škola báňská - Technická univerzita Ostrava, 2003, 218 s. ISBN 8024805030.

STMicroelectronics. STM32CubeMX [online]. [cit. 2018-04-07]. Dostupné z: http://www.st.com/en/development-tools/stm32cubemx.html

STMicroelectronics Atollic. Atollic TrueSTUDIO for STM32 [online]. [cit. 2018-04-07]. Dostupné z: https://atollic.com/truestudio/

STMicroelectronics. Getting started with STM32 Nucleo board software development tools [online] [cit. 2018-04-07]. Dostupné z: https://www.st.com/content/ccc/resource/technical/document/user_manual/1b/03/1b/b4/88/20/4e/cd/DM00105928.pdf/files/DM00105928.pdf/jcr:content/translations/en.DM00105928.pdf

STMicroelectronics. Description of STM32F4 HAL and LL drivers [online]. [cit. 2019-10-21]. Dostupné z: https://www.st.com/content/ccc/resource/technical/document/user_manual/2f/71/ba/b8/75/54/47/cf/DM00105879.pdf/files/DM00105879.pdf/jcr:content/translations/en.DM00105879.pdf

STMicroelectronics. Leading supplier of Arm® Cortex®-M microcontrollers [online]. [cit. 2018-04-07]. Dostupné z:https://www.st.com/content/ccc/resource/sales_and_marketing/promotional_material/brochure/f0/93/da/5c/6b/31/4a/96/brstm32.pdf/files/brstm32.pdf/jcr:content/translations/en.brstm32.pdf

STMicroelectronics. STM32 32-bit Arm Cortex MCUs [online] [cit. 2018-04-06] https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html

STMicroelectronics. STM32F7 Series [online] [cit. 2018-04-06] https://www.st.com/en/microcontrollers-microprocessors/stm32f7-series.html

STMicroelectronics. STM32F76xxx and STM32F77xxx advanced Arm®-based 32-bit MCUs - Reference manual [online]. [cit. 2019-10-21]. Dostupné z: https://www.st.com/content/ccc/resource/technical/document/reference_manual/group0/96/8b/0d/ec/16/22/43/71/DM00224583/files/DM00224583.pdf/jcr:content/translations/en.DM00224583.pdf

STMicroelectronics. NUCLEO-F767ZI [online] [cit. 2018-04-06] https://www.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/mcu-mpu-eval-tools/stm32-mcu-mpu-eval-tools/stm32-nucleo-boards/nucleo-f767zi.html

SUTORÝ, Tomáš. LIN (Local Interconnect Network) [online]. [cit. 2019-10-15]. Dostupné z: http://www.elektrorevue.cz/clanky/04012/index.html

ŠČEVÍK, Petr. Číslicová regulace. [online]. [cit. 2019-11-16]. Dostupné z: http://books.fs.vsb.cz/cislicovaregulace/

VÁŇA, Vladimír. Mikrokontroléry ATMEL AVR: programování v jazyce C : popis a práce ve vývojovém prostředí CodeVisionAVR C. Praha: BEN - technická literatura, 2003. ISBN 978-807-3001-025.

VÍTEČKOVÁ, Miluše a VÍTEČEK, Antonín. Základy automatické regulace. Ostrava: VŠB – Technická univerzita Ostrava, 2008 243 s. ISBN 978-80-248-1924-2.

VÍTEČKOVÁ, Miluše a VÍTEČEK, Antonín. Vybrané metody seřizování regulátorů. Ostrava: VŠB – Technická univerzita Ostrava, 2011 230 s. ISBN 978-80-248-2503-8.

VÍTEČKOVÁ, Miluše a VÍTEČEK, Antonín. Zpětnovazební řízení mechatronických systémů Ostrava: VŠB – Technická univerzita Ostrava, 2013, 200 s., ISBN 978-80-248-3232-6, Dostupné z: http://books.fs.vsb.cz/ZRMS/zpetnovazebni-rizeni-mechatronickych-systemu.pdf

WATANABE, Kento. Introduction to STM32 ARM Microcontroller with STM HAL-Library & SW4STM32 [online]. 1. Kindle, 2017 [cit. 2018-03-02].